1966-08

A comparison of steepest-ascent and second variational techniques in solving a restricted class of trajectory optimization problems.

Luders, Ernest Celestino
Stanford University

A COMPARISON OF STEEPEST-ASCENT

AND

SECOND VARIATIONAL TECHNIQUES

IN SOLVING A RESTRICTED CLASS

OF

TRAJECTORY OPTIMIZATION PROBLEMS

A THESIS

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS

AND THE COMMITTEE ON THE GRADUATE DIVISION

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

ENGINEER

By

Ernest Celestino Luders

August, 1966

PREFACE

Two feedback control schemes which maximize a terminal quantity while satisfying specified terminal conditions are discussed and compared. The schemes are based on a linear perturbation from a nominal non-optimal path which does not, in general, satisfy the terminal conditions. The methods have been programmed in the ALGOL computer language for evaluation and the programs are included in the appendices.

I would like to express my appreciation to my advisor, Dr. Benjamin O. Lange, Dr. Richard Rosenbaum of the Lockheed Missile and Space Company, and Mr. Thomas Bullock, for their guidance and assistance during the preparation of this thesis.

This work was sponsored by the United States Naval Postgraduate School, Monterey, California.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

In recent years several authors have treated the problem of determining optimum control programs for nonlinear systems with terminal constraints. These problems arise in the design of control systems and development of guidance laws where it is desired to determine, out of all possible time histories of the control variables, the one control history that maximizes (or minimizes) one terminal quantity or cost function while simultaneously yielding specified values of certain other terminal quantities.

The steepest-ascent or gradient method developed by Kelley[1], Bryson and Denham[2], which is a systematic and rapid numerical procedure, has proved to be successful in solving this class of problems. Improvement in the convergence time of the iterative process involved has been achieved by Rosenbaum[3] by a method based on the earlier work of Bryson and Denham[4].

Another successful method developed by Breakwell, et.al.[5], and modified by Bullock[6] is a second variation method in the Calculus of Variations.

The principal objectives of this thesis are to develop a simpler steepest-ascent program which will be understandable to the control engineer without a background in the Calculus of Variations and to compare the results and speed of convergence with the method developed by Bullock. In this way it is expected that the steepest-ascent program will prove to be a useful instrument in education and research, while at the same time through the comparison, illustrate the advantages and disadvantages of the two approaches to the problem.

The method used in developing the steepest-ascent program is essentially a variation of the methods of Bryson and Denham[4], and

Rosenbaum[3], hence it is restricted to problems in which only the deviations in the control variables and adjustable parameters are considered in the performance index. It is also restricted to problems in which the pay-off quantity is a function of the terminal value of the states. This variation includes a terminal-error control scheme which maintains a bound on the terminal constraint errors, hence reducing the total number of iterations required to converge to the optimum since larger deviations from the nominal trajectory can be tolerated while still meeting the desired terminal conditions.

A numerical example is given of a rocket ascent trajectory into a circular orbit of maximum altitude. Provision is made for a two-stage rocket with optimization of the inter-stage coast duration.

II. STATEMENT OF THE PROBLEM

Given a system which can be described by a set of non-linear (or linear) first order, ordinary differential equations, determine a control history $u(t)$, in the interval $t_0 \leq t \leq T$, to maximize

$$\phi = x_1(T), \tag{1}$$

subject to constraints

$$\psi = \psi[x(T)] = 0, \tag{2}$$

$$\frac{dx}{dt} = f[x(t),u(t),t], \tag{3}$$

$$t_0,\ T,\ \text{and}\ \ x(t_0)\ \ \text{given;} \tag{4}$$

where

$$u(t) = \begin{bmatrix} u_1(t) \\ \cdot \\ \cdot \\ \cdot \\ u_m(t) \end{bmatrix}, \quad \text{an } m \times 1 \text{ matrix of control variables,} \tag{5}$$

$$x(t) = \begin{bmatrix} x_1(t) \\ \cdot \\ \cdot \\ \cdot \\ x_n(t) \end{bmatrix}, \quad \text{an } n \times 1 \text{ matrix of state variables,} \tag{6}$$

$$\psi = \begin{bmatrix} \psi_1 \\ \cdot \\ \cdot \\ \cdot \\ \psi_q \end{bmatrix}, \quad \text{a } q \times 1 \text{ matrix of terminal constraint functions, each of which is a known function of } x(T), \tag{7}$$

$$f = \begin{bmatrix} f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_n \end{bmatrix}, \quad \text{an } n \times 1 \text{ matrix of known functions of } x(t),\ u(t), \text{ and } t, \text{ i.e., the system equations;} \tag{8}$$

ϕ is the pay-off quantity and is one of the states, namely $x_1(T)$; (9)

T is the terminal value of the independent variable.

On each iteration it is desired to minimize the mean value of a positive definite quadratic form in the control variable deviations:

$$C = \frac{1}{2} \int_{t_0}^{T} \delta u'(\tau)\ \delta u(\tau)\ d\tau \tag{10}$$

where the superscript (') indicates the transpose of a matrix, and $\delta u(\tau)$ are small deviations in the control history from a nominal non-optimum trajectory.

III. STEEPEST-ASCENT METHOD OF SOLUTION

A. BACKGROUND

Bryson and Denham, in Ref. 4 ~~3~~, considered terminal control of non-linear (and linear) systems for minimum mean values of a positive definite quadratic form in the control variable deviations. That is, it was assumed that a nominal control history had been determined which caused the vehicle to arrive at the terminal point with desired values of certain specified terminal conditions. Small deviations from this nominal trajectory were considered which might be caused by disturbances, inaccuracies in the data, inaccuracies in the control system, etc. The problem was to determine small deviations from the nominal control so that the terminal constraints would be satisfied in spite of the disturbances.

In the present paper the nominal trajectory is determined by guessing a reasonable control variable program. For example, in a rocket trajectory problem one might choose an initial launch angle and a gravity turn with zero thrust angle throughout as is done in the numerical example. Furthermore, it is desired not only to determine control deviations which result in meeting the terminal constraints, but also to maximize the terminal value of one of the states while minimizing a performance index. This optimization scheme is a variation of the so-called Lambda Matrix Control feedback method described in Ref. 4 ~~3~~ and the convergence method of Ref. 3 ~~2~~.

B. DERIVATION OF EQUATIONS

The optimum programming problem can be solved systematically and rapidly on a high speed digital computer using the steepest-ascent technique. As stated in (A), this technique starts by guessing a nominal control variable program, $u^*(t)$, and solving the set of differential

equations (3) with initial conditions (4), to determine a nominal trajectory. This trajectory, in general, will neither maximize ϕ nor will it satisfy the terminal constraints (2).

Consider small perturbations in the control variables, $\delta u(t)$, about the nominal, where

$$\delta u(t) = u(t) - u^*(t) \tag{11}$$

(The superscript (*) indicates terms evaluated along the nominal trajectory.) These perturbations will cause perturbations in the state variables, $\delta x(t)$, where

$$\delta x(t) = x(t) - x^*(t) \tag{12}$$

Substituting these relations into the system differential equations (3) and expanding f in a Taylor series about the nominal the result is, to first order

$$\frac{d}{dt}(\delta x) = F(t)\delta x(t) + D(t)\delta u(t) \tag{13}$$

where

$$F(t) = \left(\frac{\partial f}{\partial x}\right)^*, \quad \text{an } n \times n \text{ matrix of partial derivatives evaluated along the nominal trajectory;}$$

and

$$D(t) = \left(\frac{\partial f}{\partial u}\right)^*, \quad \text{an } n \times m \text{ matrix of partial derivatives evaluated along the nominal trajectory}$$

To determine the effects upon the terminal conditions ϕ and ψ we introduce the linear differential equations adjoint to (12) defined as

$$\frac{d\Phi}{dt}(T,t) = -\Phi(T,t)F(t) \tag{14}$$

where Φ is an n x n fundamental or state transistion matrix whose elements give the sensitivities of the terminal states to perturbations, $\delta x(t)$, along the trajectory. (See Ref. 7). Initial conditions for these equations are specified at the terminal time, i.e.,

$$\Phi(T,T) = I, \quad \text{the identity matrix} \tag{15}$$

hence numerical integration proceeds backward in time.

The solution to (14) provides a solution to the linear perturbation equations (12) at the terminal point:

$$\delta x(T) = \Phi(T,t)dx(t) + \int_t^T \Phi(T,\tau)D(\tau)\delta u(\tau)d\tau \tag{16}$$

Since ϕ and ψ depend on the terminal values of the states, small deviations, $d\phi$ and $d\psi$ may be calculated from the solution to (16). Conversely, if $\delta x(T)$ is known by specifying values of $d\phi$ and $d\psi$, the corresponding control history may be calculated, which is what is done. Rewriting (16)

$$\delta x(T) - \Phi(T,t)\delta x(t) - \int_t^T \Phi(T,\tau)D(\tau)\delta u(\tau)d\tau = 0 \tag{17}$$

In order to minimize the performance index subject to the constraints (16), the method of Lagrange multipliers is employed. Multiplying (16) by a matrix of Lagrange multipliers

$$\nu = \begin{bmatrix} \nu_1 \\ \cdot \\ \cdot \\ \cdot \\ \nu_{q+1} \end{bmatrix}, \tag{18}$$

(where (16) is written to include only those states which appear in ϕ and ψ, hence q+1 equations) and adjoining the result to (10)

$$C = \nu'\delta x(T) - \nu'\Phi(T,t)\delta x(t) + \int_t^T \left[\frac{1}{2}\,\delta u'(\tau)\delta u(\tau) - \nu'\Phi(T,\tau)D(\tau)\delta u(\tau)\right] d\tau \tag{19}$$

Completing the square

$$C = \nu'[\delta x(T) - \Phi(T,t)\delta x(t)] + \int_t^T \left[\frac{1}{2}\,(\delta u(\tau)D'(\tau)\Phi'(T,\tau)\nu)'(\delta u(\tau) - D'(\tau)\Phi'(T,\tau)\nu) - \right.$$

$$\left. - \frac{1}{2}\,\nu'\Phi(T,\tau)D(\tau)D'(\tau)\Phi'(T,\tau)\nu\right] d\tau \tag{20}$$

To minimize C subject to the control variations choose

$$\delta u(\tau) = D'(\tau)\Phi'(T,\tau)\nu \tag{21}$$

Substituting this relation in (17) in order to find ν

$$\delta x(T) - \Phi(T,t)\delta x(t) - \int_t^T \Phi(T,\tau)D(\tau)D'(\tau)\Phi(T,\tau)\nu\, d\tau = 0 \tag{22}$$

Define the controllability matrix

$$J = +\int_t^T \Phi(T,\tau)D(\tau)D'(\tau)\Phi'(T,\tau)\, d\tau \tag{23}$$

Note: This integration may be performed simultaneously with (14) by numerically solving

$$\frac{dJ}{dt} = -\Phi(T,t)D(t)D'(t)\Phi'(T,t) \tag{24}$$

with initial conditions

$$J(T) = 0 \tag{25}$$

Equation (22) may be written

$$\delta x(T) - \Phi(T,t)\delta x(t) = J\nu \tag{26}$$

Solving for ν

$$\nu = J^{-1}[\delta x(T) - \Phi(T,t)\delta x(t)] \tag{27}$$

Substituting this relation in (21)

$$\delta u(t) = D'(t)\Phi'(T,t)J^{-1}[\delta x(T) - \Phi(T,t)\delta x(t)] \tag{28}$$

which is the perturbation in the control history which satisfies the constraint (17) while minimizing the performance index (10).

As stated earlier, $\delta x(T)$ is determined from the specified values of $d\varphi$ and $d\psi$ where

$$d\varphi = \delta x_1(T) \tag{29}$$

and

$$d\psi = \psi[\delta x(T)] \tag{30}$$

As yet, nothing has been said as to how one chooses the desired payoff improvement, $d\varphi$, or the desired improvement in meeting the terminal constraints, $d\psi$. The latter is normally chosen as

$$d\psi = -\psi \tag{31}$$

that is, the negative of the total error on any iteration is chosen as the desired correction specified on the following iteration. The problem of specifying $d\varphi$ is more complex and is the subject of the next section.

It is worthy to note at this point that the Lagrange multipliers, which are error feedback terms, need not be computed at every point on the trajectory. Sufficient accuracy can be obtained in computing the control deviations (21) by solving (27) at discrete intervals and using the result until the next "sampling time". This reduces the number of

times the controllability matrix must be inverted per iteration and materially improves the running time of the program. Experimentation will reveal how large the sampling interval can be made. Since the controllability matrix is singular at the terminal time, T, new values of the Lagrange multipliers should not be computed too close to the end of the trajectory.

C. METHOD OF SPECIFYING THE IMPROVEMENT IN PAY-OFF

In general, one does not know how far from the optimum a given nominal trajectory will be. It is, therefore, difficult to guess how much pay-off improvement to specify initially. However, it is possible to compute a value of $d\varphi$ which will result in a trajectory that satisfies the terminal constraints. This is done as follows:

The changes in the control variables required to meet the terminal constraints with the pay-off unconstrained are given by

$$\delta u(t) = \Lambda' \left[\int_{t_0}^{T} \Lambda \, \Lambda' d\tau \right]^{-1} d\psi \tag{32}$$

where

$$\Lambda = \Phi(T,t)D(t), \text{ without row 1 or column 1.}$$

This is the same as the basic control equation (28) with $\delta x(t_0)$ equal to zero and $\delta x(T)$ containing only the terminal constraint terms.

The change in pay-off, $d\varphi$, that will be produced by a given change in the control variables is, from (16)

$$d\varphi = \delta x_1(T) = \int_{t_0}^{T} \Lambda_1 \delta u(\tau) d\tau \tag{33}$$

where Λ_1 is the first row of the Λ matrix. Substituting (33) in (32)

where Λ_1 is the first row of the Λ matrix. Substituting (33) in (32)

$$d\varphi = \delta x_1(T) = \left[\int_{t_0}^{T} \Lambda_1 \; \Lambda' d\tau\right]\left[\int_{t_0}^{T} \Lambda \; \Lambda' d\tau\right]^{-1} d\psi \tag{34}$$

Equation (34) gives the change in pay-off associated with adjusting the control in order to meet the terminal constraints. This value is used on the first iteration.

Equation (34) is also used to compute a value of the pay-off corrected for terminal errors, i.e., $\varphi + d\varphi$ is the value the pay-off would have achieved had the terminal errors been zero. Hence, one can determine whether an improvement in the pay-off was actually achieved or if an apparent improvement was a result of larger terminal constraint errors.

On subsequent iterations, one of three methods is used to compute $d\varphi$. A value equal to 25 per cent of the nominal value of φ is computed and stored. This quantity is called $d\varphi^{**}$ and is used in method (2) below. It is a fairly arbitrary choice but should be made as large as seems reasonable. The program will automatically adjust it if it is too large.

Method 1. Choose $d\varphi$ to satisfy the terminal constraints with the pay-off unconstrained as described above.

Method 2. If $|d\psi| \leq \epsilon$, where ϵ is chosen as reasonable tolerance on the terminal constraints then

$$d\varphi = \frac{d\varphi^{**}}{2^i} \tag{35}$$

where i is a count of the number of times method (3) has failed. This has the effect of halving the improvement specified each time a run is unsuccessful in improving the terminal errors or the corrected value of the pay-off. The program

terminates when, while executing this method, $d\varphi$ becomes less than a pre-set number. A final run is then made using method (1).

Method 3. If $|d\psi| > \epsilon$ the following questions are asked: Were the errors on the current iteration smaller than on the preceeding iteration? Was there an improvement in the corrected value of the pay-off? If the answer to either question is no, the run is considered unsuccessful, the control history is replaced by the previous control history, and method (1) is used. If the answer to either question is yes, $d\varphi$ is set equal to zero and an attempt is made to satisfy $|d\psi| \leq \epsilon$. If this test fails a second time, method (1) is used.

D. TWO-STAGE ROCKET TRAJECTORY WITH COAST PERIOD OPTIMIZATION

In many orbit injection applications, such as the Gemini-Titan II system, the launch vehicle is made up of two powered stages. It is therefore of interest to consider the effect of an interstage coast phase on the maximum altitude obtainable. In this section a method of calculating the optimum coast duration is derived.

The basic equations, (1) through (12), are the same. The linear perturbation equations (13) may be written

$$\frac{d}{dt}[\delta x(t)] = F(t)\delta(t) + D(t)\delta u(t) + B(t)\delta c \qquad (36)$$

where

$$B(t) = \left(\frac{\partial f}{\partial c}\right)^{*}, \quad \text{an n x 1 matrix of partial derivatives of f with respect to the coast duration, c.} \qquad (37)$$

The solution to (37) is

$$\delta x(T) = \Phi(T,t)\delta x(t) + \int_t^T \Phi(T,\tau)D(\tau)\delta u(\tau)d\tau + \int_t^T \Phi(T,\tau)B(\tau)\delta c d\tau \tag{38}$$

The performance index becomes

$$C = \frac{1}{2}\int_t^T \delta u'(\tau)\delta u(\tau)d\tau + \frac{1}{2}\lambda\, \delta c^2 \tag{39}$$

where λ is simply a weighting factor.

Before attempting to minimize (39) subject to the constraint (38), a method must be derived to evaluate the last term in (38) which is the change in the terminal values of the states due to a change in the coast duration. Since c is an adjustable parameter which does not appear explicitly in f, the partial derivatives cannot be evaluated directly. However the desired term may be calculated by the following method:

Define

t_1 = Stage I burnout time

t_2 = Stage II ignition time

hence

$$c = t_2 - t_1 \tag{40}$$

Since δc is a small time increment we may write

$$x(t_2 + \Delta t) - x(t_2) = \int_{t_2}^{t_2+\Delta t} \frac{dx}{d\tau}\, d\tau$$

$$= \int_{x(t_2)}^{x(t_2)+\Delta x}$$

$$= (x(t_2) + \Delta x) - x(t_2);$$

consequently we may make the approximation

$$x(t_2 + \delta c) - x(t_2) \cong \left(\frac{dx}{dt}\right)_{t_2} \delta c \tag{41}$$

Now define x_u as the states evaluated with the thrust off (uncontrolled), and x_c as the states evaluated with the thrust on (controlled). From (41)

$$x_u(t_2 + \delta c) = x(t_2) + \dot{x}_u(t_2)\delta c$$

$$x_c(t_2 + \delta c) = x(t_2) + \dot{x}_c(t_2)\delta c$$

where $(\cdot)$ indicates differentiation with respect to time. Subtracting the above expressions we have

$$x_u(t_2 + \delta c) - x_c(t_2 + \delta c) = [\dot{x}_u(t_2) - \dot{x}_c(t_2)]\delta c$$

Define

$$\delta x_c = [\dot{x}_u(t_2) - \dot{x}_c(t_2)]\delta c \tag{42}$$

The quantity δx_c is a perturbation in the states occurring at time

$$t = t_2 + \delta c$$

due to cutting off the thrust for a period δc. The question remains: How does this perturbation propagate to the end of the trajectory? The answer is clearly

$$\delta x(T) = \Phi(T, t_2 + \delta c)\delta x_c \tag{43}$$

Finally, (38) may be written

$$\delta x(T) = \Phi(T,t)\delta x(t) + \int_t^T \Phi(T,\tau)D(\tau)\delta u(\tau)d\tau + \Phi(T,t_2 + \delta c)\delta x_c \tag{44}$$

where the last term is zero prior to time $t_2 + \delta c$.

Introducing the Lagrange multipliers, ν, and adjoining (44) to (39)

$$C = \frac{1}{2}\int_t^T \delta u'(\tau)\delta u(\tau)d\tau + \frac{1}{2}\lambda\delta c^2 + \nu'\delta x(T) - \delta'\Phi(T,t)\delta x(t) \tag{45}$$

$$- \nu'\int_t^T \Phi(T,\tau)D(\tau)\delta u(\tau)d\tau - \nu'\Phi(T,t_2 + \delta c)\delta x_c$$

As before, completing the square yields the optimum control change

$$\delta u(\tau) = D'(\tau)\Phi'(T,\tau)\nu \tag{46}$$

By differentiation the optimum coast change is

$$\delta c = \frac{1}{\lambda}\nu'\ \Phi(T,t_2 + \delta c)[\dot{x}_u(t_2) - \dot{x}_c(t_2)] \tag{47}$$

Substituting (46) and (47) into (44)

$$\delta x(T) = \Phi(T,t)\delta x(t) + \int_t^T \Phi(T,\tau)D(\tau)D'(\tau)\Phi'(T,\tau)d\tau\nu$$

$$+ \frac{1}{\lambda}\nu'\Phi(T,t_2 + \delta c)[\dot{x}_u(t_2) - \dot{x}_c(t_2)][\dot{x}_u(t_2) - \dot{x}_c(t_2)]'\ \Phi'(T,t_2 + \delta c)\nu$$

Define

$$\Delta = \delta x(T) - \Phi(T,t)\delta x(t) \tag{49}$$

$$J = \int_t^T \Phi(T,\tau)D(\tau)D'(\tau)\Phi(T,\tau)d\tau \tag{50}$$

$$A = [\dot{x}_u(t_2) - \dot{x}_c(t_2)] \tag{51}$$

$$G = \Phi(T,t_2 + \delta c)A\ A'\ \Phi'(T,t_2 + \delta c) \tag{52}$$

Rewriting (48)

$$\Delta = J\nu + \frac{1}{\lambda}G\nu \tag{53}$$

$$\nu = \left(J + \frac{G}{\lambda}\right)^{-1}\Delta \tag{54}$$

Finally, substituting in (46) and (47), the control and coast variations are

$$\delta u(t) = D'(\tau)\Phi'(T,\Phi) \left(J + \frac{G}{\lambda}\right)^{-1} \Delta \tag{55}$$

$$\delta c = \frac{1}{\lambda} A'\Phi(T,t_2 + \delta c)\left(J + \frac{G}{\lambda}\right)^{-1} [\delta x(T) - \Phi(T,t_2 + \delta c)\delta x(t_2 + \delta c)] \tag{56}$$

Since the last term in (44) is zero prior to time $t_2 + \delta_c$, the term G/λ in (55) and (56) is also zero prior to that time.

Equation (56) should be evaluated at $t = t_2 + \delta c$, but since δc is the unknown, it is evaluated at t_2. This does not introduce an appreciable error if δc is small.

In some numerical integration procedures the terminal value of the independent variable cannot be changed once the integration has begun, hence the change in coast time cannot be added immediately. This problem is solved by evaluating (56) on each forward integration (except the nominal) and, if $\delta c \neq 0$, re-integrating the latter portion of the trajectory from t_1 to T.

E. COMPUTATIONAL PROCEDURES

As stated earlier, this steepest-ascent technique requires the use of a high speed digital computer. The sequence of operations is summarized here.

1. Compute the nominal path by integrating the system differential equations (3) with a nominal control history and appropriate initial conditions and store the time history of the state variables at reasonably small intervals. Print out the values of φ and ψ.

2. Integrate the adjoint differential equations (14) backward, evaluating the partial derivatives on the nominal path by reference to the states stored in step (1). Simultaneously integrate the controllability matrix equations (24). Store the results at the same interval as the states.

3. Select desired terminal condition changes, $d\varphi$ and $d\psi$, as explained in Sections B and C.

4. Compute and use the new control history while integrating the system differential equations forward. Again print out the values of φ and ψ, unless the next step applies.

5. If the two-stage rocket problem is being solved, compute the new coast period in step (4). Transfer the storage locations of the second stage control history to correspond with the new coast time. Integrate the system equations from t_1 to the new terminal time. Print out the values of φ and ψ.

6. Repeat procedures (2) through (5) until the pay-off improvement in step (3) is less than a preset value. At this point, use method (1) described in Section C to select $d\varphi$ and complete step (4) and (5). This has the effect of eliminating any remaining errors in the terminal constraints.

7. Punch cards or store the control history on tape and terminate the program.

Before concluding this section, a few general factors of great importance in this type of numerical calculation should be discussed.

The programmer must exercise great care when working with values of type real (or floating point). Often a calculation is made where the result is expected to be an integral value such as $4/2 = 2.000 \ldots$, however, due to the binary, octal, and decimal conversions which take place within the computer, the result may come out 1.9999...99. This problem occurs when trying to generate array storage indices based on a value of the independent variable which is a floating point quantity.

IV. SECOND VARIATION METHOD OF SOLUTION

A. OUTLINE OF THE METHOD

Bullock[6] has derived a feedback control scheme based on the second order variational theory in the Calculus of Variations. The method is outlined here in sufficient detail to solve the problem stated in II.

The differential equations to be satisfied are

$$\dot{x} = f(x, u, t) \tag{57}$$

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial x}\right)' \lambda \tag{58}$$

$$\begin{pmatrix} \dot{M} \\ \dot{N} \end{pmatrix} = \begin{pmatrix} F & -Q \\ -S & -F' \end{pmatrix} \begin{pmatrix} M \\ N \end{pmatrix} \tag{59}$$

and

$$\dot{b} = M'd - N'c \tag{60}$$

in order to maximize

$$\varphi = \varphi[x(T), T] \tag{61}$$

and satisfy the terminal constraints

$$\psi = \psi[x(T), T] \tag{62}$$

Define the variational Hamiltonian

$$H = \lambda' f \tag{63}$$

The elements of Eqs. (59) and (60) are

$$F = f_x - f_u H_{uu}^{-1} H_{ux} \tag{64}$$

$$Q = f_u H_{uu}^{-1} f' \tag{65}$$

$$S = H_{xx} - H_{xu} H_{uu}^{-1} H_u' \tag{66}$$

$$c = - f_u H_{uu}^{-1} H_u' \tag{67}$$

$$d = H_{xu} H_{uu}^{-1} H_u' . \tag{68}$$

where the subscripts indicate partial differentiation in the usual sense.

The initial conditions for (57) are given and the terminal conditions for (58), (59), and (60) are

$$\lambda^{(T)} = (\varphi_x - \nu'\psi_x)_{t=T} \tag{69}$$

where the components of the column vector ν are sensitivities of the pay-off φ to changes in the terminal constraints ψ;

$$M(T) = I - \psi_x'(\psi_x \psi_x')^{-1} \psi_x \tag{70}$$

where I is the identity matrix;

$$N(T) = -\psi_x'\psi_x \tag{71}$$

$$b(T) = -\psi_x \, d\psi \tag{72}$$

where $d\psi = -\psi$.

The perturbations in the control history are given by

$$\delta u(t) = u^* - u = -H_{uu}^{-1}(H_u + H_{ux}\delta x + f_u' \delta\lambda) \tag{73}$$

where

$$\delta\lambda = (M')^{-1}(N'\delta x + b). \tag{74}$$

In order to minimize a performance index

$$C = \frac{1}{2} \int_t^T \delta u'(\tau)\delta u(\tau)d\tau \tag{75}$$

H_{uu} in the above equations is modified by adding an arbitrarily large negative constant, K, which is reduced in magnitude as the program converges. This has the effect of constraining the magnitude of δu.

Since the terminal conditions on the adjoint equations, $\lambda(\tau)$, depend initially on the choice of ν, a method is given which will improve the accuracy of these terminal conditions on subsequent iterations.

Equation (74) is an expression for $\delta\lambda$ at any time, t, but since M is singular at T, it cannot be evaluated directly. However, if a point (t_1) is chosen sufficiently far from this singularity, the following equation can be integrated from t_1 to T:

$$\delta\dot{\lambda} = -S\delta x - F'\,\delta\lambda \tag{76}$$

with initial conditions

$$\delta\lambda(t_1) = (M')^{-1}(N'\,\delta x + b)\Big|_{t=t_1} \tag{77}$$

The solution to (76) is then added to the current values of $\lambda(T)$ prior to the next backward integration of (58).

B. COMPUTATIONAL PROCEDURES

As in the steepest-ascent method, this method requires the use of a digital computer. The sequence of operations is summarized here.

1. Select a nominal control history and initial values of ν. This can be done by starting with a control program and ν generated by the Steepest-ascent method, where

$$\nu = \left(\int_t^T \Lambda_1 \Lambda' \, d\tau \right) \left(\int_t^T \Lambda \Lambda' \, d\tau \right)^{-1}$$

from Eq. (35).

2. Integrate the system differential equations as in the steepest-ascent method.

3. Integrate Eqs. (58), (59), and (60), backward with the appropriate terminal conditions. If the determinant of M changes sign or H_{uu} becomes positive, store the current value of the time and stop the integration. The reason for this is explained below.

4. From Eq. (73), compute the new control history while integrating the system equations forward.

5. Compare the value of φ and ψ obtained to those obtained on the nominal trajectory. If the pay-off, φ, or the terminal constraints, ψ, have become worse the run is considered unsuccessful and a tighter bound is placed on δu by increasing the magnitude of K. If, on the other hand, φ and ψ are the same or have improved, the run was successful and (3) and (4) are repeated.

6. The program is terminated when no change occurs in the pay-off or the constraints and $|K| << |H_{uu}|$. At this point the control history is stored on punched cards or tape.

7. If in step (3) the determinant of M changed sign (this condition is called a "conjugate point" in the Calculus of Variations), or H_{uu} became positive (which indicates the Legengre condition is not satisfied), the integration in (4) is begun at a slightly later time than this condition occurred. Normally, on subsequent iterations, this point will move backward to the beginning of the trajectory and disappear.

V. PROGRAM EXAMPLES

A single-stage rocket trajectory problem as described below was programmed utilizing each of the methods discussed. The ALGOL computer language was used and the programs were run on a Burroughs B-5500 digital computer.

Assuming the rocket is launched from an airless, non-rotating Earth, the state equations are

$$\dot{x}_1 = \dot{r} = V \sin(\gamma)$$

$$\dot{x}_2 = \dot{\theta} = \frac{V}{r} \cos(\gamma)$$

$$\dot{x}_3 = \dot{V} = g_0 \left(\frac{T}{W_0}\right) \left(\frac{\cos(u)}{1 - \frac{T}{W_0} \frac{t}{Isp}}\right) - \frac{\mu \sin(\gamma)}{r^2}$$

$$\dot{x}_4 = \dot{\gamma} = \frac{g_0}{V} \left(\frac{T}{W_0}\right) \left(\frac{\sin(u)}{1 - \frac{T}{W_0} \frac{t}{Isp}}\right) - \frac{\mu \cos(\gamma)}{r^2 V} + \frac{V \cos(\gamma)}{r}$$

where r = altitude measured from the center of the Earth, V = velocity, γ = flight path angle, g_0 = gravitational acceleration at the Earth's surface, T = thrust (assumed constant), u = thrust angle (measured from the velocity vector), W_0 = initial weight, Isp = specific impulse, t = time, μ = universal gravitational constant, θ = downrange angle. The initial conditions are

$r(0) = R_e$ (Earth radius)

$\theta(0) = 0$

$V(0) = 100$ ft/sec

$\gamma(0) = 89.87$ degrees

It is desired to place the payload in a circular orbit of maximum altitude, hence

$$\varphi = x_1(T) = r(T)$$

$$\psi = \begin{pmatrix} x_2 - \sqrt{\frac{\mu}{r}} \\ x_4 \end{pmatrix}_{t=T} = 0$$

Appendices (A) and (B) contain listings of the steepest-ascent program and second variation program respectively. Comments are inserted at strategic points which explain the sequence of operations.

The steepest-ascent program contains logic for a single or dual stage rocket. It was run in the single stage mode to generate a nominal control history for input to the second variation program and to compare results. It was also run in the two-stage mode to test the coast optimization logic.

The input data for the steepest-ascent program are

1. Initial velocity (feet/second) (must be non-zero).
2. Launch angle (degrees).
3. Duration of the first stage burn (seconds), for single stage rockets this quantity is the total burn time.
4. First stage thrust (pounds).
5. Second stage thrust (pounds), for single stage rockets zero is input.
6. First stage fuel flow rate (pounds/second).
7. Second stage fuel flow rate (pounds/second), for single stage, any non-zero number.
8. Rocket liftoff weight (pounds).

9. Second stage weight after separation (pounds), for single stage, any non-zero number.

10. Initial value of coast duration (seconds), for single stage, zero.

11. Duration of the second stage burn (seconds), for single stage, zero.

12. Coast weighting factor, λ.

13. Number of stages (1 or 2).

The input data for the second variation program are:

1. Initial velocity (feet/second).
2. Launch angle (degrees).
3. Duration of rocket burn (seconds).
4. Thrust divided by initial weight (pound/pound).
5. ν_1
6. ν_2
7. Integration step size and data storage interval.
8. K (See Section IV-A).
9. Nominal control history.

Other parameters which the user may desire to change must be changed inside the program or incorporated into the READ statement.

For the single-stage runs, the following input values were used:

Initial velocity................100 ft/sec

Launch angle....................89.87 degrees

Duration of burn................220 seconds

Thrust..........................430,000 pounds

Fuel flow rate..................1433.3 pounds/second

Liftoff weight..................333,770 pounds

ν_1.............................678,914

ν_2............................. -93.58

Step size......................2 seconds

K..............................Several values

For the two-stage runs, data for the Gemini-Titan II system were used:

Initial velocity................100 feet/second

Launch angle....................89.87 to 89.95 degrees

First stage burn................150 seconds

First stage thrust..............430,000 pounds

Second stage thrust.............100,000 pounds

First stage fuel flow...........1666.6 pounds/second

Second stage fuel flow..........327.7 pounds/second

Liftoff weight..................331,000 pounds

Second stage weight.............70,000 pounds

Coast duration..................10 seconds

Second stage burn...............180 seconds

Coast weighting factor..........0.1

Number of stages................2

VI. RESULTS AND DISCUSSION

A. STEEPEST-ASCENT VS. SECOND VARIATION

When the steepest-ascent program was run in the single-stage mode, the nominal trajectory attained an altitude of 196,015 feet. The errors in meeting the terminal constraints on the velocity and flight path angle were 426 feet per second and 1.146 degrees. On the third iteration the altitude was improved to 260,427 feet with terminal errors of 0.67 feet per second and 0.025 degrees. The program was terminated when the desired altitude change, $d\varphi$, became less than 5,000 feet. At this point fifteen iterations had been completed. The terminal altitude achieved was 318,126 feet with terminal errors of 0.64 feet per second and 0.003 degrees. The associated control history was punched on cards and values of ν_1 and ν_2 were printed out. The program ran five minutes and nine seconds. It is estimated that this time would be about halved if the program were run on an IBM 7090 computer.

The output generated in the steepest-ascent program was used as input to the second variation program. As expected, the nominal trajectory attained an altitude 318,126 feet. On the succeeding forward integration the trajectory was totally unreasonable. The control deviations were made smaller by increasing the initial magnitude of K but this failed to improve the results. Small variations were made in the input values of ν which caused relatively large changes in the results but it was not clear how to make adjustments which would improve the performance of the program. The running time was far in excess of the steepest-ascent program, taking over three minutes to compile and complete just one iteration.

B. TWO STAGE ROCKET TRAJECTORY WITH COAST OPTIMIZATION

As stated in Section IV, the input data for this problem were those of the Gemini-Titan II launch system with an arbitrary choice of ten seconds for the initial coast duration. Lambda, the coast weighting factor, was chosen as 0.1 since earlier runs indicated that a value of 1.0 caused the coast variations to be insignificant. This choice proved to be satisfactory.

The nominal trajectory attained an altitude of 392,564 feet with terminal errors of 25 feet per second in velocity and 0.84 degrees in flight path angle. On the first iteration, where the program attempts only to meet terminal constraints, the altitude was improved by 18,018 feet, the terminal errors were 0.09 feet per second and 0.0025 degrees. On the fourth iteration the coast time was reduced to eight seconds. At this point, a new nominal for the portion of the trajectory following first stage burnout was computed using the new coast period. The result of this change in coast was an improvement in the velocity constraint of 12 feet per second, and a degradation of the flight path angle by 0.25 degrees. The terminal altitude achieved on this iteration was 573,450 feet. On the fifth iteration the coast period was reduced to two seconds, this accounted for an altitude improvement of 30,973 feet. On the sixth iteration the coast period was reduced to zero, this improved the terminal altitude by 13,477 feet. In both instances cited above where the coast period was changed, the terminal constraint errors were diminished.

The altitude improvement specified on the second iteration was 50,000 feet. The program was allowed to run until this figure was reduced to less than 1000 feet. This proved to be rather wasteful as the terminal altitude

failed to improve significantly after the desired improvement was reduced to 6,250 feet, which occurred on the eleventh iteration. In Fig. 1 the terminal altitude, corrected for terminal constraint errors, achieved on each iteration is illustrated. It clearly shows that twelve iterations were sufficient to converge to the optimum.

Figure 2 illustrates the convergence of the control history. The discontinuities which occurred are due to using discrete feedback at twenty second intervals rather than continuous feedback. The curves are of different lengths due to the changes in coast period which occurred.

Figure 3 illustrates the action of the terminal error control scheme. Each time the errors became excessive they were reduced to essentially zero in one iteration.

The last terminal altitude achieved was 616,573, a 57.2 per-cent improvement over the initial nominal.

C. SENSITIVITY OF THE STEEPEST-ASCENT PROGRAM TO CHANGES IN INITIAL CONDITIONS

The sensitivity of the steepest-ascent program was tested to determine the capabilities of the convergence scheme. Lauch angles of 89.87 and 89.95 degrees were tested on the single stage trajectory. The resulting terminal altitudes were 196,015 and 803,428 feet. In the latter case the terminal errors were 1858 feet per second in velocity, and 40.4 degrees in flight path angle. In both cases the program converged in twelve iterations to approximately the same optimum altitude.

An attempt was made to solve the problem given a ninety degree launch angle which failed. Further testing at lower launch angles was not accomplished due to computer time limitations, however, it is believed that the tests conducted amply illustrate the virtue of the method.

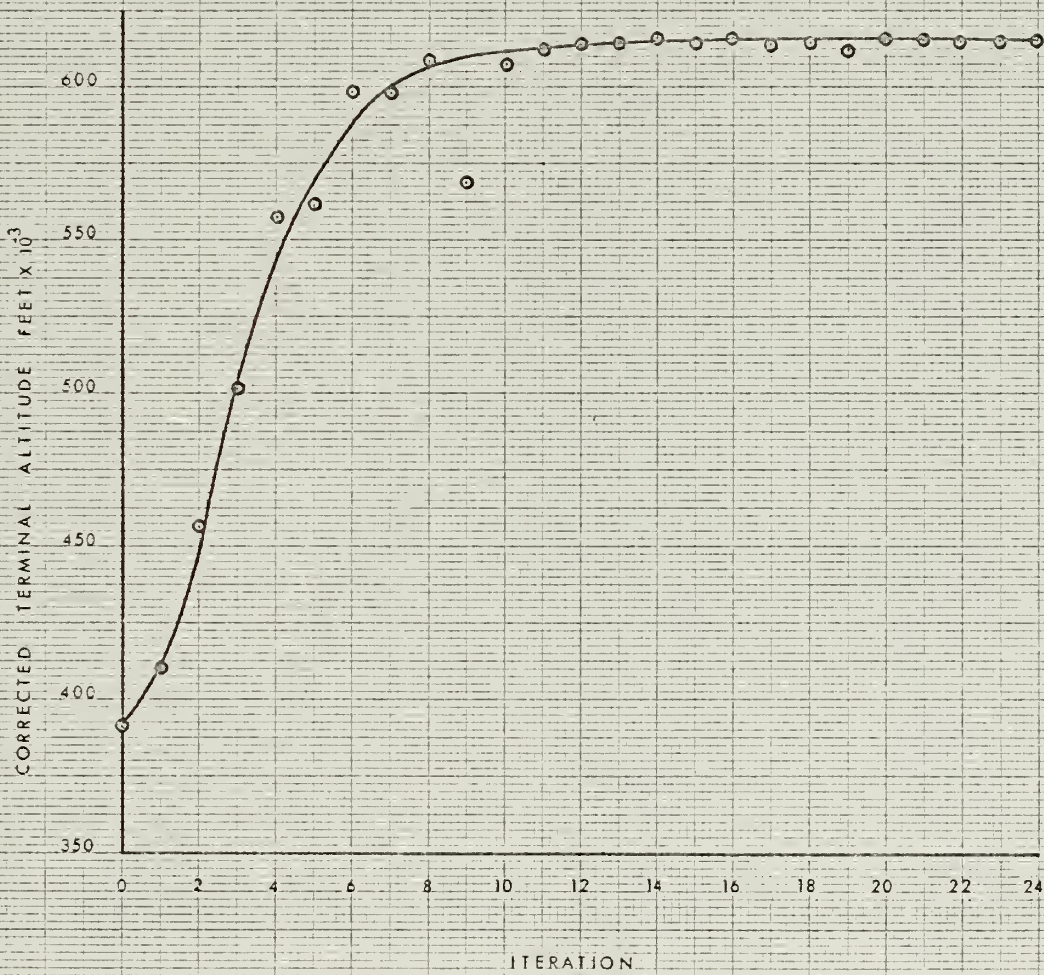


Fig. 1. Terminal altitude achieved on each iteration.

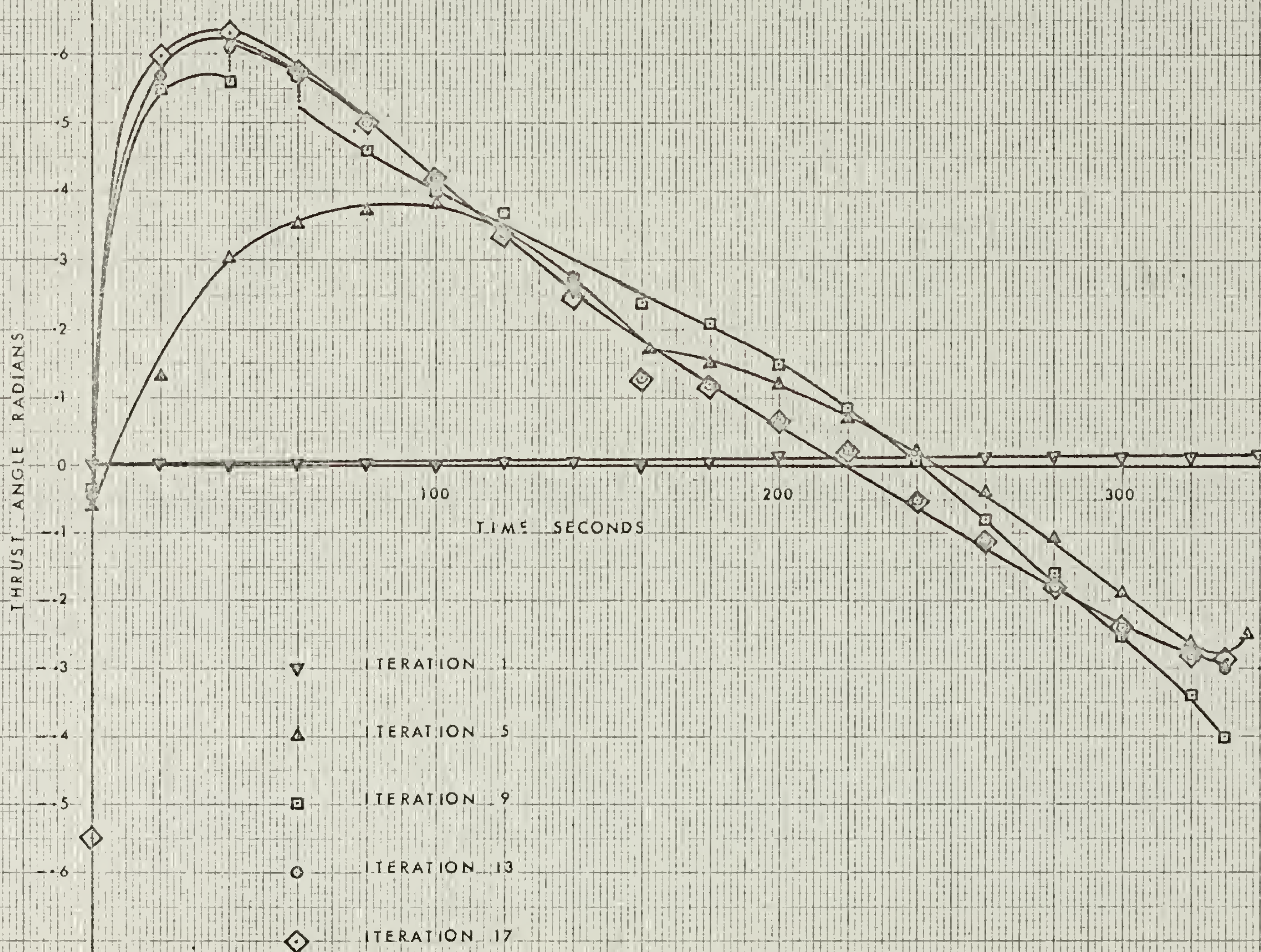
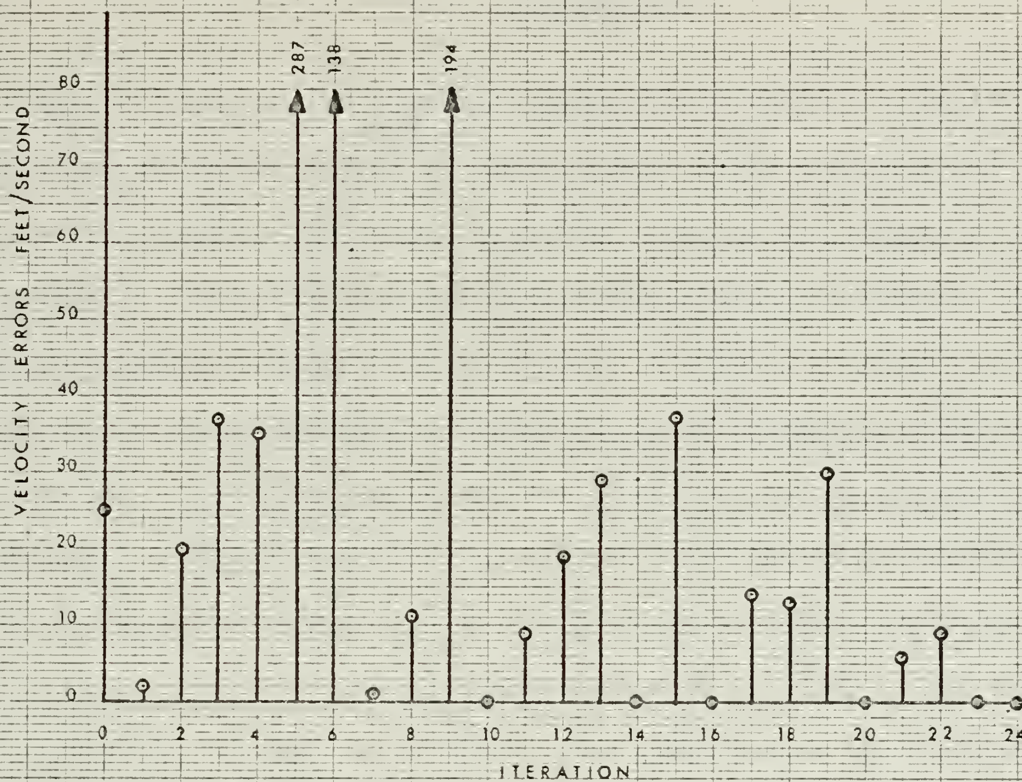


Fig. 2. Convergence of the control history.



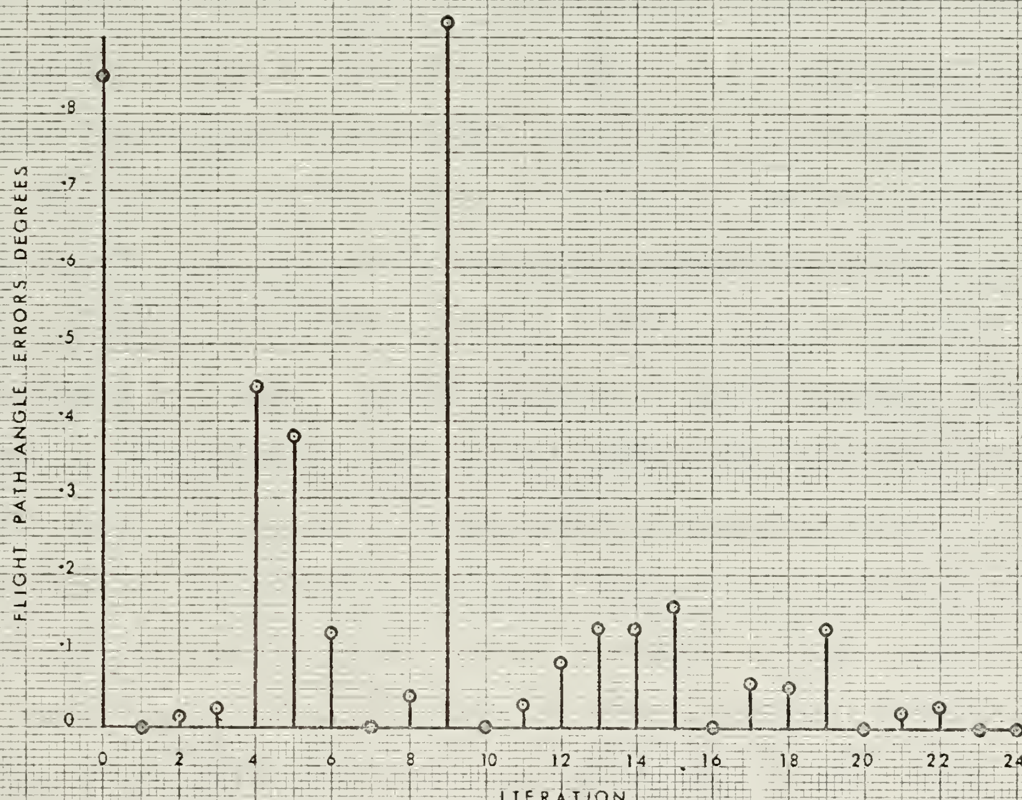


Fig. 3. Illustration of the terminal error control.

VII. CONCLUSIONS

The terminal control feedback scheme due to Bryson and Denham[4] and the method due to Rosenbaum[3] of leaving the pay-off unconstrained in order to satisfy terminal constraints have been combined and altered as necessary to produce a simplified steepest-ascent optimization procedure. This procedure has been shown to be successful in solving a typical rocket trajectory problem including optimization of an interstage coast period.

In this closed-loop procedure the change in control is computed at each point utilizing continuous or discrete information on the deviation from the previous trajectory. It is closed loop because the program continuously (or periodically) checks on how it is doing in its attempt to satisfy the terminal constraints. The advantage of this procedure is that larger deviations from the nominal can be tolerated while still maintaining a bound on the terminal errors. It is, therefore, possible to move rapidly toward the optimum trajectory as is illustrated in Fig. 2.

The second variation method due to Bullock[6] has been applied to the same problem with questionable results. Consultation between the author and Mr. Bullock has failed to uncover possible flaws in the theory or the programming technique. Bullock has shown in several examples that the method is successful, however, in each example the Hamiltonian did not depend explicitly on time. In the rocket problem this is not the case. Although this fact has no theoretical bearing on the problem, it is the only major difference between Bullock's examples and this problem.

Experimentation revealed that the rocket problem is extremely sensitive to the choice of ν. This was not the case in the examples presented by Bullock. It was thought that the values of ν generated by

the steepest-ascent program would be very close to correct, however, in view of the failure, this premise was laid open to question. Hence, one reasonable conclusion that can be drawn is that ν was chosen incorrectly and that there is no intuitive or analytical method presently available to make the proper choice. There exists, of course, the possibility of error, but considerable time and painstaking effort has been expended to minimize this possibility.

Failure of the second-variation method notwithstanding, some conclusions may be drawn with respect to the advantages and disadvantages of the two methods.

1. Understanding the theory involved in the second-variation method requires a background in the Calculus of Variations, whereas the steepest-ascent method presented does not.

2. The second variation method apparently requires a good estimate of the sensitivities, ν, while the steepest-ascent method only requires a guess of the nominal control and will tolerate a fairly poor guess.

3. The second variation method requires backward integration of the $2n^2 + 2n$ equations M, N, b, and λ (where n is the number of state variables) and a matrix inversion at every step of the forward integration. The steepest-ascent method requires backward integration of Φ and J which is less than $2n^2$ equations since J is symmetric. The matrix inversion can be done at less frequent intervals using the sampled data feedback method suggested. The second variation method thus requires more computer time per iteration.

VIII. ADAPTATION OF THE PROGRAMS TO OTHER TYPE PROBLEMS

An attempt was made to generalize each program so that they could be easily adapted to other problems. This required a large number of subscripted variables and matrix multiplication loops in the subprograms containing the differential equations. Since the subprograms are called twice per integration step they must be as efficient as possible. The use of subscripted variables and loops is most inefficient and results in more than doubling the running time.

Adaptation to another problem is still relatively simple however. The programs contain sufficient comments to indicate where necessary changes must be made.

APPENDIX A. STEEPEST-ASCENT COMPUTER PROGRAM

```
BEGIN COMMENT ERNEST C. LUDERS  BOX 219  STEEPEST-ASCENT;

    INTEGER FLAG,FAIL,COUNT,QUIT;
    REAL T1,COAST,FF1,FF2,ISP1,ISP2,WR1,WR2,W1,W2,T2,MM1,MM2,
         RE, MASRAT, TOVERWO, GO, K, ISP, TF, HH, SAMPLETIME,
         VO,PREDHFO,PREHFO,DHFO,HFO,J12,J13,PREVF,PREGAMF,
         PREHF, DVF, DGAMF, BOUND, BOUND2, DET, EX, CNR, P,
         MM3,LAMDA,SUM,OLDCOAST,TB2,G,DHF;

    INTEGER MAXINDEX,L,M,I,ITER,STAGES,OLDMAXINDEX,
           COMPUTECOAST,COASTCOMPUTE;

     REAL ARRAY D[1:3,1:3], A[1:3], YP[0:4,0:110];

    SAVE ARRAY XP,OLDXP[0:4,0:200],LP[0:22,0:200], ERR[0:3],
             TEMP[0:4], DEL[0:3], CHK[1:3,1:3],
             TMP[1:2,1:2], PHI ,OLDPHI[0:200],JEI,
             JINV[1:3,1:3];

    LABEL L1, L2, L3, L4, L5,L6,HELL;

    FORMAT F1 (//, X25, "AT TIME T = ", I3, "  THE DETERMI",
             "NANT OF J = ",
       E18.11, //, X33, 3E23.11, //, X18, "CHECK MATRIX = ",
       3E23.11,//, X33, 3E23.11, //),
    F2 (X20, "THIS RUN REQUESTED AN ADDITIONAL:", //),
    F3 (X27,F12.2, "       FEET OF ALTITUDE", //, X30, F9.2,
       "       FEET/SEC OF VELOCITY", //, X32, F11.6,
```

```
   "  DEGREES OF FLIGHT PATH ANGLE", //),
F4 (X20, "THE RUN ACTUALLY YIELDED AN ADDITIONAL:", //),
F5 (X20, "TERMINAL ALTITUDE                         =", F10.2,
   "        FEET", /, X20,  "TERMINAL VELOCITY",X17,"=",
   X1, F9.2, "        FEET/SEC", /, X20,
   "TERMINAL FLIGHT PATH ANGLE          =    ", F11.6,
   "   DEGREES", /,
   X20, "ORBITAL VELOCITY AT THIS ALTITUDE =", X1, F9.2,
   "        FEET/SEC"),
F6 (X20,"COAST DURATION",X20,"=",F10.2,"         SECONDS"),
F7 (X20,"CORRECTED TERMINAL ALTITUDE          =",F10.2,X7,
   "FEET"),
   HISTORY(4F18.12);

DEFINE LOOPL = FOR L←1,2,3,4 DO #;

FILE OUT CARDS 0 (2,10);

COMMENT  MATRIX MULTIPLICATION PROCEDURE GOES HERE;

COMMENT  MATRIX INVERSION PROCEDURE GOES HERE;

COMMENT  THIS PROCEDURE CONTAINS THE SYSTEM DIFFERENTIAL
EQUATIONS AND IS USED ON THE FIRST NOMINAL, ON RUNS IN
WHICH THE COAST TIME HAS BEEN CHANGED , AND ON THE FINAL
PRECISION RUN;
```

```
    PROCEDURE FUNCT(T, X, F);
    VALUE T;  REAL T;  REAL ARRAY X, F[1];
    BEGIN REAL R, V, S, C,FEE,TS;INTEGER I, J;
COMMENT INTERPOLATE IN THE CONTROL ARRAY FOR THE
PROPER VALUE;
        I ← ENTIER(T/HH);  J ← IF I<MAXINDEX THEN I+1 ELSE I;
        FEE ← (Q←PHI[I]) + (T/HH - I)×(PHI[J] - Q);
        R ← X[1] + RE;
        S ← SIN(X[4]);  C ← COS(X[4]);  V ← X[3];
TS←T-T2; COMMENT TIME FROM STAGE TWO IGNITION;
IF T<T1 OR STAGES =1 THEN BEGIN
      TOVERWO←FF1/W1;
      MASRAT←1-TOVERWO×T/ISP1;
              END ELSE TOVERWO←0;
IF T≥T2 AND T2≠TF THEN BEGIN
      TOVERWO←FF2/W2;
      MASRAT←1-TOVERWO×TS/ISP2; END;
COMMENT THESE ARE THE SYSTEM DIFFERENTIAL EQUATIONS;
        F[1] ← V×S;
        F[2] ← V×C/R;
        F[3] ← (Q ← G0×TOVERWO/MASRAT)×COS(FEE) - K×S/R*2;
        F[4] ← -K/V×C/R*2 + Q×SIN(FEE)/V + F[2];
    END FUNCT;

COMMENT  THIS IS THE BACKWARD INTEGRATION PROCEDURE WHICH
CONTAINS THE ADJOINT EQUATIONS AND THE CONTROLLABILITY
MATRIX EQUATIONS;
```

```
     PROCEDURE LINBAK (TB, LS, LF);    VALUE TB;
     REAL TB;  REAL ARRAY LS, LF[1];
     BEGIN REAL R, V, S, C, T, SF, L1, L2, L3, L4, INT;
     INTEGER I,J;
    REAL TS;
    REAL L5,L6,L7,L8,N1,N2,N3,N4,N5,N6,N7,N8;
     REAL TSM,TCM,M1,M2,M3,FEE,L13,L14,L33,L34,L43,L44;
COMMENT T IS BACKWARD RUNNING TIME;
        T ← TF - TB;  I ← ENTIER(T/HH);
        IF I≤0 THEN I ← 0;
        J ← IF I<MAXINDEX THEN I+1 ELSE I;  INT ← T/HH - I;
        L1←LS[1];  L2←LS[2];  L3←LS[3];  L4←LS[4];
        L5←LS[5];  L6←LS[6];  L7←LS[7];  L8←LS[8];
        N1←LS[9];  N2←LS[10]; N3←LS[11]; N4←LS[12];
        N5←LS[13]; N6←LS[14]; N7←LS[15]; N8←LS[16];
COMMENT INTERPOLATE FOR THE PROPER VALUES OF THE STATES;
        R ← (Q←XP[1,I]) + INT×(XP[1,J] - Q) + RE;
        V ← (Q←XP[3,I]) + INT×(XP[3,J] - Q);
        S ← SIN((Q←XP[4,I]) + INT×(XP[4,J] - Q));
        C ← COS((Q←XP[4,I]) + INT×(XP[4,J] - Q));
        SF ← SIN((Q←PHI[I]) + INT×(PHI[J] - Q));
TS←T-T2; COMMENT TIME TO GO UNTIL STAGE TWO IGNITION;
IF T≥T2 AND T2≠TF THEN BEGIN
      TOVERWO←FF2/W2;
      MASRAT←1-TOVERWO×TS/ISP2; END
                    ELSE TOVERWO←0;
IF T<T1 OR STAGES =1 THEN BEGIN
```

```
   TOVERWO←FF1/W1;
   MASRAT←1-TOVERWO×T/ISP1;
END;
COMMENT THESE ARE THE ADJOINT DIFFERENTIAL EQUATIONS;
    LF[1] ←¯V×C×L2/R*2 +2×K×S×L3/R*3 - C×(V - 2×K/(V×R))×L4/
          R*2;
    LF[2] ← 0;
    LF[3] ← +S×L1 +C×L2/R +C×(1 + K/(V*2×R))×L4/R - GO×
          TOVERWO×SF/(V*2×MASRAT)×L4;
    LF[4] ← +V×C×L1 - V×S×L2/R -K×C×L3/R*2 - S×(V - K/(V×R))
          ×L4/R;
  LF[5 ] ← ¯V×C×L6/R*2 +2×K×S×L7/R*3 - C×(V - 2×K/(V×R))×L8/
          R*2;
    LF[6] ← 0;
  LF[7 ] ←  +S×L5 +C×L6/R +C×(1 + K/(V*2×R))×L8/R - GO×
          TOVERWO×SF/(V*2×MASRAT)×L8;
  LF[8 ] ←  +V×C×L5 - V×S×L6/R -K×C×L7/R*2 - S×(V - K/(V×R))
          ×L8/R;
  LF[9 ] ← ¯V×C×N2/R*2 +2×K×S×N3/R*3 - C×(V - 2×K/(V×R))×N4/
          R*2;
  LF[10] ←  0;
  LF[11] ←  +S×N1 +C×N2/R +C×(1 + K/(V*2×R))×N4/R - GO×
          TOVERWO×SF/(V*2×MASRAT)×N4;
  LF[12] ←  +V×C×N1 - V×S×N2/R -K×C×N3/R*2 - S×(V - K/(V×R))
          ×N4/R;
  LF[13] ← ¯V×C×N6/R*2 +2×K×S×N7/R*3 - C×(V - 2×K/(V×R))×N8/
          R*2;
  LF[14] ←  0;
```

```
LF[15] ←  +S×N5 +C×N6/R +C×(1 + K/(V*2×R))×N8/R - GO×
          TOVERWO×SF/(V*2×MASRAT)×N8;
LF[16] ←  +V×C×N5 - V×S×N6/R -K×C×N7/R*2 - S×(V - K/(V×R))
          ×N8/R;
FEE ← (Q←PHI[I]) + INT×(PHI[J]-Q);
     TSM ← GO×TOVERWO/MASRAT×SIN(FEE);
     TCM ← GO×TOVERWO/MASRAT×COS(FEE)/V;
 M1 ← -L3×TSM + L4×TCM;
 M2 ← -N3×TSM + N4×TCM;
 M3 ← -N7×TSM + N8×TCM;
COMMENT THESE ARE THE CONTROLLABILITY MATRIX EQUATIONS;
    LF[17] ← M1*2; LF[18]← M2*2; LF[19]← M3*2;
    LF[20]← M1×M2; LF[21]← M2×M3; LF[22]← M3×M1;
END LINBAK;


COMMENT  THIS IS THE FORWARD INTEGRATION PROCEDURE WHICH
COMPUTES THE NEW CONTROL;


   PROCEDURE SLOPE (T, X, F);    VALUE T;
   REAL T;  REAL ARRAY X, F[1];
   BEGIN REAL R, V, S, C, TB, IND, BKIND, FEE, TSM, TCM, M1,
           M2, M3, TS;
     LABEL L1, L2, L3, L4, TCOCLOSE,L5,L6;    INTEGER I, J;
TS←T-T2; COMMENT TIME FROM STAGE TWO IGNITION;
IF T<T1 OR STAGES =1 THEN BEGIN
    TOVERWO←FF1/W1;
    MASRAT←1-TOVERWO×T/ISP1; ISP←ISP1;
```

```
            END ELSE TOVERWO←0;
IF T≥T2 AND T2≠TF THEN BEGIN
     TOVERWO←FF2/W2; ISP←ISP2;
     MASRAT←1-TOVERWO×TS/ISP2; END;
       IF T≥BOUND×.999999999 OR COMPUTECOAST=1 THEN
       BEGIN
COMMENT THIS IS THE SAMPLED DATA FEEDBACK SECTION AND IS
HIT AT THE SAMPLING INTERVAL AND AT T2 WHEN A NEW  COAST
PERIOD IS COMPUTED;
          IF COMPUTECOAST≠1 THEN
          BEGIN
             IND←BOUND/HH;
             BOUND←BOUND+SAMPLETIME;
          END ELSE IND←ENTIER(T2/HH+.0000001);
          BKIND←MAXINDEX-IND;  SUM←0;
COMMENT  SINCE J GETS SINGULAR NEAR TF, SKIP THIS SECTION
IF T IS CLOSE TO TF;
          IF BOUND ≥ TF-30  THEN GO TO TOOCLOSE;
          FOR I←1,2,3 DO JEI[I,I] ← JINV[I,I]←LP[I+16,BKIND]+
          (IF T≥T2×.999999 OR COMPUTECOAST=1 THEN D[I,I]/
          LAMDA ELSE 0);
          JEI[1,2]←JINV[1,2]←JEI[2,1]←JINV[2,1]←LP[20,BKIND]+
          (IF T≥T2×.999999 OR COMPUTECOAST=1 THEN D[1,2]/
          LAMDA ELSE 0);
          JET[2,3]←JINV[2,3]←JEI[3,2]←JINV[3,2]←LP[21,BKIND]+
          (IF T≥T2×.999999 OR COMPUTECOAST=1 THEN D[2,3]/
          LAMDA ELSE 0);
          JEI[1,3]←JINV[1,3]←JEI[3,1]←JINV[3,1]←LP[22,BKIND]+
```

```
        (IF T≥T2×.999999 OR COMPUTECOAST=1 THEN D[1,3]/
        LAMDA ELSE 0);
        IF COMPUTECOAST=1 THEN COMPUTECOAST←0;
        P ← TIME(1);
        INVERT(JINV,3,I);
        IF I=1 THEN BEGIN
        WRITE(<"THE J MATRIX IS SINGULAR AT TIME = ",
                F6.2>,T);
COMMENT IF THE J MATRIX BECOMES SINGULAR TERMINATE THE RUN;
        GO TO HELL END;
        LOOPL TEMP[L] ← OLDXP[L,IND];
        LOOPL  SUM ← SUM + LP [L, BKIND]×(X[L] - TEMP[L]);
        DEL[1] ← DHF - SUM;  SUM ← 0;
        LOOPL  SUM ← SUM+LP[L+8,  BKIND]×(X[L] - TEMP[L]);
        DEL[2] ← DVF - SUM;  SUM ← 0;
        LOOPL  SUM ← SUM+LP[L+12, BKIND]×(X[L] - TEMP[L]);
        DEL[3] ← DGAMF - SUM;  SUM ← 0;
        ERR[1] ← ERR[2] ← ERR[3] ← 0;
        FOR L←1,2,3 DO FOR M←1,2,3 DO
        ERR[L] ← ERR[L] + JINV[L, M]×DEL[M];
COMMENT  ERR IS THE MATRIX OF LAGRANGE MULTIPLIERS, THIS
IS THE END OF THE SAMPLED DATA FEEDBACK SECTION;
TOOCLOSE: END;
COMMENT  THE CONTROL IS COMPUTED BETWEEN HERE AND L4, NOTE
THAT THE CONTROL IS COMPUTED ONE STEP AHEAD OF THE CURRENT
TIME TO PERMIT INTERPOLATION;
 L1:   IF T≥BOUND2×.999999999 THEN
       BEGIN IF T=0 THEN BEGIN I←0;  GO TO L2 END;
```

```
L3:      I ← BOUND2/HH + 1;  BOUND2 ← BOUND2 + HH;
         IF BOUND2>TF THEN GO TO L4;
L2:      FEE ← PHI[I];  V ← XP[3,I];  BKIND ← MAXINDEX - I;
         TS←BOUND2;
         IF BOUND2≥T2 THEN TS←BOUND2-T2;
         TSM←G0×TOVERWO/(Q←(1-TOVERWO×TS/ISP))×SIN(FEE);
         TCM ← G0×TOVERWO/Q×COS(FEE)/V;
         M1 ← -TSM×LP [3,BKIND] + TCM×LP [4,BKIND];
         M2 ← -TSM×LP[11,BKIND] + TCM×LP[12,BKIND];
         M3 ← -TSM×LP[15,BKIND] + TCM×LP[16,BKIND];
COMMENT COMPUTE THE NEW VALUE OF THE CONTROL;
         PHI[I] ← (OLDPHI[I]←PHI[I]) +M1×ERR[1] +M2×ERR[2]
                   + M3×ERR[3];
         IF I=ENTIER(T2/HH+.0000001)+1 AND STAGES=2 THEN
         COMPUTECOAST←1
         ELSE COMPUTECOAST←0;
         IF T=0 AND I=0 THEN GO TO L3 END;
 L4:     I ← ENTIER(T/HH);  J ← IF I<MAXINDEX THEN I+1 ELSE I;
       FEE ← (Q←PHI[I]) + (T/HH - I)×(PHI[J] - Q);
       R ← X[1] + RE;
       S ← SIN(X[4]);  C ← COS(X[4]);  V ← X[3];
COMMENT HERE ARE THE SYSTEM DIFFERENTIAL EQUATIONS;
       F[1] ← V×S;
       F[2] ← V×C/R;
       F[3] ← (Q ← G0×TOVERWO/MASRAT)×COS(FEE) - K×S/R*2;
       F[4] ← -K/V×C/R*2 + Q×SIN(FEE)/V + F[2];
COMMENT IN THIS SECTION THE NEW COAST DURATION IS COMPUTED;
       IF COMPUTECOAST=1 OR COASTCOMPUTE=1 THEN GO TO L5
```

```
        ELSE GO TO L6;
L5:     COASTCOMPUTE←COASTCOMPUTE+1;
        A[1]←0;
        A[2]←-K×S/R*2 - F[3];
        A[3]←-K/V×C/R*2 + F[2] - F[4];
        BKIND←MAXINDEX-ENTIER(T2/HH+.0000001);
        MM1←A[2]×LP[3,BKIND] + A[3]×LP[4,BKIND];
        MM2←A[2]×LP[11,BKIND] + A[3]×LP[12,BKIND];
        MM3←A[2]×LP[15,BKIND] + A[3]×LP[16,BKIND];
        D[1,1] ← MM1*2; D[2,2] ← MM2*2; D[3,3] ← MM3*2;
        D[1,2] ← D[2,1] ← MM1×MM2;
        D[1,3] ← D[3,1] ← MM1×MM3;
        D[2,3] ← D[3,2] ← MM2×MM3;
        IF COMPUTECOAST=1 THEN GO TO L6;
        COAST ← (OLDCOAST←COAST)+(Q←(MM1×ERR[1]+MM2×ERR[2]
                 +MM3×ERR[3])/LAMDA);
COMMENT  COAST IS SET TO THE NEAREST INTEGER DIVISIBLE BY
THE STORAGE INTERVAL, THIS AIDS IN KEEPING INDICES STRAIGHT;
        WRITE(<"DELTACOAST = ",E20.10>,Q);
        COAST←ENTIER((COAST+.9)/2)×2;
        IF COAST<0 OR STAGES =1 THEN COAST←0;
        WRITE(<"   COAST = ",F20.10>,   COAST);
L6: END SLOPE;


COMMENT  INTEGRATION PROCEDURE GOES HERE;


COMMENT MAIN PROGRAM BEGINS HERE;
COMMENT INITIALIZE BELLS, FLAGS, AND CONSTANTS;
```

```
    ITER←0;  FLAG←0;  FAIL←1; COUNT←0; QUIT←0;
    RE ← 2.0987;  G0 ← 32.17;  K ← G0×RE*2;
COMMENT  SAMPLETIME IS THE FEEDBACK SAMPLING INTERVAL AND
HH IS THE DATA STORAGE INTERVAL,  HH IS ALSO THE INTEGRATION
STEP SIZE;
                    SAMPLETIME ← 20 ;   HH ←  2 ;
L1: READ(XP[3,0],XP[4,0],T1,FF1,FF2,WR1,WR2,W1,W2,COAST,TB2,
        LAMDA,STAGES)[L4];
COMMENT THE INPUT DATA IS
            XP[3,0] = INITIAL VELOCITY
            XP[4,0] = LAUNCH ANGLE
            T1 = FIRST STAGE BURN TIME;
COMMENT     FF1 = FIRST STAGE THRUST
            FF2 = SECOND STAGE THRUST
            WR1 = FIRST STAGE FUEL FLOW RATE;
COMMENT     WR2 = SECOND STAGE FUEL FLOW RATE
            W1 = LAUNCH WEIGHT
            W2 = SECOND STAGE WEIGHT AFTER SEPARATION;
COMMENT     COAST = INITIAL CHOICE OF THE COAST DURATION
            TB2= SECOND STAGE BURN TIME
            LAMDA = COAST WEIGHTING FACTOR;
COMMENT     STAGES = NUMBER OF STAGES;
    OLDCOAST←COAST;
    T2←T1+COAST;
    TF←T2+TB2;
    OLDMAXINDEX←MAXINDEX ← ENTIER(TF/HH+.51);
    ISP1←FF1/WR1;  ISP2←FF2/WR2;
    XP[1,0] ← XP[2,0] ← 0;  XP[4,0] ← XP[4,0]/57.2957795131;
```

```
COMMENT  GENERATE OR READ IN THE NOMINAL CONTROL HISTORY;
    FOR I←0 STEP 1 UNTIL MAXINDEX DO PHI[I] ← 0;
COMMENT  INITIAL CONDITIONS FOR BACKWARD INTEGRATION;
    FOR I←2 STEP 1 UNTIL 22 DO LP[I,0]←0;
    FOR I←1 STEP 5 UNTIL 16 DO LP[I,0]←1;
COMMENT  COMPUTE THE NOMINAL TRAJECTORY;
    ADAMS(4, HH, 0, TF, HH, TF, 0, 0, XP, FUNCT);
COMMENT  PERFORM THE BACKWARD INTEGRATION;
L2: ADAMS(22,HH, 0, TF, HH, TF, 0, 0, LP , LINBAK);
COMMENT  STORE THE OLD VALUES OF THE STATES BEFORE COMPUTING
A NEW FORWARD TRAJECTORY;
    FOR I←0 STEP 1 UNTIL MAXINDEX DO LOOPL OLDXP[L,I] ← XP[L,I];
COMMENT  BETWEEN HERE AND L3 THE ALTITUDE CHANGE DUE TO
TERMINAL ERRORS IS COMPUTED;
    PREHF ← XP[1,MAXINDEX];  PREVF ← XP[3,MAXINDEX];
    PREGAMF ← XP[4,MAXINDEX];
          J12←LP[20,MAXINDEX];
          J13←LP[22,MAXINDEX];
          TMP[1,1]←LP[18,MAXINDEX];
          TMP[2,1]←TMP[1,2]←LP[21,MAXINDEX];
          TMP[2,2]←LP[19,MAXINDEX];
          INVERT(TMP,2,I);
IF I=1 THEN BEGIN WRITE(<"THE LAMBDA  MATRIX IS SINGULAR">);
          GO TO L4 END;
          TMP[1,1]←J12×TMP[1,1]+J13×TMP[2,1];
          TMP[1,2]←J12×TMP[1,2]+J13×TMP[2,2];
          VO ← SQRT(K/(PREHF+RE));
          PREDHFO←TMP[1,1]×(VO-PREVF)-TMP[1,2]×PREGAMF;
```

```
        PREHFO←PREHF+PREDHFO;
        WRITE(<"OLD CORRECTED HF = ",F20.10>,PREHFO);
L3: IF FLAG =0 THEN BEGIN
      DHF←PREDHFO;
        DVF←SQRT(K/(PREHF+DHF+RE))-PREVF;
        DGAMF←-PREGAMF; END;
    BOUND ← BOUND2 ← COASTCOMPUTE ← COMPUTECOAST ← 0;
COMMENT  COMPUTE THE NEW CONTROL WHILE INTEGRATING THE
SYSTEM EQUATIONS FORWARD;
    ADAMS(4, HH, 0, TF, HH, TF, 0, 0, XP, SLOPE);
COMMENT  THE FOLLOWING LOGIC CAUSES THE CONTROL HISTORY
TO BE PRINTED OUT EVERY FOURTH ITERATION;
    IF ENTIER(ITER/4)×4 = ITER THEN BEGIN
      WRITE(FOR I←0 STEP 1 UNTIL MAXINDEX DO PHI[I]);
      WRITE([PAGE]) END;
    ITER←ITER+1;
    OLDMAXINDEX←MAXINDEX;
    IF COAST≠OLDCOAST THEN
    BEGIN

COMMENT  ADJUST STORAGE LOCATIONS OF PHI AND FLY NEW
NOMINAL TO PROVIDE PROPER INPUT OF STATES FOR BACKWARD
INTEGRATION;

      T2←T1+COAST;
      TF←T2+TB2;
      OLDMAXINDEX←MAXINDEX;
      MAXINDEX←ENTIER(TF/HH+.51);
```

```
      L←ENTIER((COAST-OLDCOAST)/HH+.0000001);
      FOR I←ENTIER((T1+OLDCOAST)/HH+.0000001) STEP 1
      UNTIL OLDMAXINDEX DO BEGIN
         PHI[I+L]←PHI[I];
         IF I<ENTIER(T2/HH+.0000001) -1 THEN PHI[I+1]←0;
      END;
      I←ENTIER(T1/HH+.0000001);
      LOOPL  YP[L,0]←XP[L,I];
      ADAMS(4, HH, T1, TF, HH, TF,  0,  0, YP, FUNCT);
      FOR M←I  STEP 1 UNTIL MAXINDEX DO
      LOOPL XP[L,M]←YP[L,M-I];
      END;
COMMENT  PRINT OUT THE RESULTS OF THE ITERATION;
    WRITE(F2);  WRITE(F3, DHF, DVF,  DGAMF× 57.295779513);
    WRITE(F4);  WRITE(F3, XP[1,MAXINDEX] - PREHF,
      XP[3,MAXINDEX] - PREVF, (XP[4,MAXINDEX] - PREGAMF)
      ×57.295779513);
    WRITE (F5, XP[1,MAXINDEX], XP[3,MAXINDEX],
      XP[4,MAXINDEX]×57.295779513, SQRT(K/(RE +
      XP[1,MAXINDEX])));
    WRITE(F6,   COAST);
COMMENT  COMPUTE THE CORRECTED TERMINAL ALTITUDE;
         VO←SQRT(K/(XP[1,MAXINDEX]+RE));
         DHFO←TMP[1,1]×(VO-XP[3,MAXINDEX])-
              TMP[1,2]×XP[4,MAXINDEX];
         HFO←XP[1,MAXINDEX] + DHFO; .
COMMENT PRINT OUT THE CORRECTED TERMINAL ALTITUDE;
         WRITE(F7,HFO);
```

```
         IF QUIT=1 THEN GO TO L6;
COMMENT  TEST TERMINAL ERRORS WITHIN TOLERANCE;
      IF ABS(DVF←(VO-XP[3,MAXINDEX]))>50
         OR ABS(DGAMF←(XP[4,MAXINDEX])) > .02
         OR SIGN(DHF)≠SIGN(XP[1,MAXINDEX]-PREHF) THEN
       BEGIN
         FAIL←FAIL+1;
          COUNT←COUNT+1;
COMMENT  TEST FOR IMPROVEMENT IN SATISFYING THE TERMINAL
CONSTRAINTS OR IMPROVEMENT IN THE CORRECTED TERMINAL ALTITUDE;
         IF (ABS(DVF)≤ABS(SQRT(K/(PREHF+RE))-PREVF) AND
         ABS(DGAMF)≤ABS(PREGAMF)) OR (HFO > PREHFO) THEN
             BEGIN
                FLAG←1; DHF←(IF COUNT=1 THEN 0 ELSE DHFO);
                GO TO L5 END ELSE BEGIN FLAG←0;
         FOR I←0 STEP 1 UNTIL (MAXINDEX←OLDMAXINDEX) DO
         BEGIN COMMENT IF THE ITERATION WAS UNSUCCESSFUL
               DISCARD THE DATA AND ATTEMPT ONLE TO SATISFY
               TERMINAL CONSTRAINTS;
         PHI[I]←OLDPHI[I];
         LOOPL XP[L,I]←OLDXP[L,I];
         END;
         COAST←OLDCOAST;
         T2←T1+COAST;
         TF←T2+TB2;
         IF ITER=1 THEN BEGIN
            WRITE([PAGE]);
COMMENT  IF THE FIRST ITERATION IS UNSUCCESSFUL, THE
```

```
FOLLOWING MESSAGE IS PRINTED OUT;
            WRITE(<"THE TERMINAL CONSTRAINTS HAVE BEEN VIO",
                   "LATED EXCESSIVELY BECAUSE",//,"THE NO",
                   "MINAL CONTROL OR THE INITIAL CONDITIO",
                   "NS ARE",//,"NOT CLOSE ENOUGH TO THE ",
                   "OPTIMUM.....GUESS AGAIN">);GO TO L4
            END;
         GO TO L3 END;
         END
       ELSE BEGIN FLAG←1; DHF←100000/(2*FAIL); COUNT←0;
COMMENT  THE FOLOWING STATEMENT CONTROLS PROGRAM TERMINATION;
          IF DHF≤1000 THEN BEGIN DHF←DHFO;QUIT←1; GO TO L5
                             END;
       END;
L5:      DVF←SQRT(K/(XP[1,MAXINDEX]+DHF+RE))-XP[3,MAXINDEX];
         DGAMF←-XP[4,MAXINDEX];
         GO TO L2 ;
COMMENT  FINAL PRECISION RUN;
L6:       ADAMS(4, 1, 0, TF, 20. 20, @-4, @-4, XP, FUNCT)  ;
COMMENT PRINT OUT SENSITIVITIES OF PAY-OFF TO ERRORS IN
TERMINAL CONSTRAINTS;
     WRITE(TMP[1,1],TMP[1,2]);
COMMENT  PUNCH THE OPTIMUM CONTROL HISTORY ON CARDS;
HELL:WRITE(CARDS,HISTORY, FOR I←0 STEP 1 UNTIL MAXINDEX DO
           PHI[I]);
    GO TO L1;      L4:  END.
```

APPENDIX B. SECOND VARIATION COMPUTER PROGRAM

```
BEGIN COMMENT  ERNEST C. LUDERS  BOX 219  SECOND VARIATION;
    REAL  RE,G0,K,ISP,TF,TOVERWO,NU1,NU2,HH,V0,TE1,TE2,
          R,V,TSING,OLDDETM,DETM,OLDTE1,OLDTE2,HUK,T3,T4,
          OLDNU1, OLDNU2, OLDT;
      REAL  SUM,FEE,MASRAT,S,C,Q,C1,C2,C3,C4,INT;
     REAL ARRAY TEMP3,TEMP6[1:3],TEMP4,TEMP5[1:3,1:3],
                  DELX,TEMP1, TEMP2,B[1:3],MINV,M,N[1:3,1:3],
                  CO[0:4,0:115];
      INTEGER  IND, BKIND;
    INTEGER  ITER,MAXINDEX,I,ONCE,KK,J,L,ST;
    INTEGER BOUND;
    REAL ARRAY OLDXP,XP,YP[0:4,0:125],PHI,HU,HUU[0:125],
              FU,HUX[1:3,0:125], ZZ[0:24,0:125],
              FL,SL[1:3,1:3,100:110],ZP[0:3,0:1];
    LABEL  L1,L2;
FORMAT HISTORY(4F18.12);
       FORMAT  F1("ON THIS ITERATION",//,X5,"NU1 = ",E20.11,
                  //,X5,
                 "NU2 = ",E20.11,//,X5,"TE1 = ",E20.11,//,
                 X5,"TE2 = ",
                 E20.10,//,X5,"HUK = ",E20.11);
    DEFINE  LOOPI = FOR I←1,2,3 DO #, LOOPJ = FOR J←1,2,3 DO #,
            LOOPL = FOR L←1,2,3 DO #;


COMMENT  MATRIX INVERSION PROCEDURE GOES HERE;
PROCEDURE NOMINAL (T,X,DX);
    VALUE T;  REAL T;  REAL ARRAY X,DX[1];
```

```
    BEGIN
       REAL S,C,MASRAT,FEE,Q;
       INTEGER I,J;
       I ← ENTIER(T/HH);  J ← IF I<MAXINDEX THEN I+1 ELSE I;
       FEE ← (Q←PHI[I]) + (T/HH - I)×(PHI[J] - Q);
       R←X[1]+RE;
       MASRAT←1-TOVERWO×T/ISP;
       S←SIN(X[4]);  C←COS(X[4]);  V←X[3];
       DX[1]← V×S;
       DX[2]← V×C/R;
      DX[3] ← (Q ← GO×TOVERWO/MASRAT)×COS(FEE) - K×S/R*2;
      DX[4] ← -K/V×C/R*2 + G×SIN(FEE)/V +DX[2];
END NOMINAL;

PROCEDURE BACK(T,Z,DZ);

    VALUE T; REAL T;  REAL ARRAY Z,DZ[1];

    BEGIN
       REAL  TB,KS,A,LAM1,LAM2,LAM3,KC,R2,R3,V2,S,C,SF,CF,
             MASRAT,U,INT;
       REAL ARRAY FX,HXX,F,Q,SS,M,N,SM,FTN,FM,QN[1:3,1:3],CC,
            D,MTD,NTC,HXU[1:3];
       INTEGER E;
       LABEL EN;
       TB←TF-T;  I←ENTIER(TB/HH+.000000001);
       IF I≤0 THEN I ← 0;  MASRAT←1-TOVERWO×TB/ISP;
       J ← IF I<MAXINDEX THEN I+1 ELSE I;  INT ←TB/HH - I;
```

```
R ← (U←XP[1,I]) + INT×(XP[1,J] - U) + RE;
V ← (U←XP[3,I]) + INT×(XP[3,J] - U);
S ← SIN((U←XP[4,I]) + INT×(XP[4,J] - U));
C ← COS((J←XP[4,I]) + INT×(XP[4,J] - U));
SF ← SIN((U←PHI[I]) + INT×(PHI[J] - U));
CF ← COS((U←PHI[I]) + INT×(PHI[J] - U));


COMMENT  COMPUTE PARTIAL DERIVATIVES OF SYSTEM EQUATIONS
WITH RESPECT TO STATES;

FX[1,1]←0;  FX[1,2]←S;  FX[1,3]←V×C;
FX[2,1]←2×K×S/R*3;  FX[2,2]←0;  FX[2,3]←-K×C/R*2;
FX[3,1]←-V×C/R*2+2×K×C/(V×R*3);
FX[3,2]←C/R+K×C/(V×R)*2-GO×TOVERWO×SF/(V*2×MASRAT);
FX[3,3]←-V×S/R+K×S/(V×R*2);

COMMENT  COMPUTE PARTIAL DERIVATIVES OF SYSTEM EQUATIONS
WITH RESPECT TO CONTROL;

FU[1,I]←0;
FU[2,I]←-(A←GO×TOVERWO/MASRAT)×SF;
FU[3,I]←A/V×CF;


COMMENT  COMPUTE DERIVATIVES OF THE HAMILTONIAN;

LAM1←Z[22];  LAM2←Z[23];  LAM3←Z[24];
```

```
        HU[I]←LAM2×FU[2,I]+LAM3×FU[3,I];
        HUU[I]←-LAM2×A×CF-LAM3×A×SF/V - HUK;

COMMENT  CHECK HUU ≥ 0;

        IF HUU[I]≥0 THEN
           IF TSING=0 THEN
           BEGIN
              TSING←TB+HH×5;
              WRITE(<"HUU ≥ 0 AT TIME = ",F20.10>,TB);
              WRITE(<"HUU = ",E20.10>,HUU[I]);
              IF TSING≥TF THEN
              BEGIN
                 TSING←0;
                 TSING←1/TSING
              END;
              GO TO EN
           END
           ELSE GO TO EN;
        HUX[1,I]←HUX[3,I]←HXU[1]←HXU[3]←0;
        HXU[2]←HUX[2,I]←-LAM3×FU[3,I]/V;
        KS←K×S;  KC←K×C;  R2←R×R;  V2←V×V;  R3←R2×R;
        HXX[1,1]←-6×LAM2×KS/R2*2+LAM3×(-6×KC/(R2×R2×V)+2×V×C/R3);
        HXX[1,2]←HXX[2,1]←-LAM3×(2×KC/(V2×R3) + C/R2);
        HXX[1,3]←HXX[3,1]←2×LAM2×KC/R3+LAM3×(-2×KS/(R3×V)+V×S/R2);
        HXX[2,2]←-LAM3×(-2×A×SF/(V2×V)+2×KC/(R2×V2×V));
        HXX[2,3]←HXX[3,2]←LAM1×C + LAM3×(-KS/(R2×V2)-S/R);
        HXX[3,3]←-LAM1×V×S + LAM2×KS/R2 + LAM3×(KC/(R2×V)-V×C/R);
```

```
KK←0;
LOOPL  LOOPJ
   BEGIN
      F[L,J]←FX[L,J]-FU[L,I]×HUX[J,I]/HUU[I];
      Q[L,J]←FU[L,I]×FU[J,I]/HUU[I];
     SS[L,J]←HXX[L,J]-HXU[L]  ×HXU[J]  /HUU[I];
      M[L,J]←Z[(KK←KK+1)];  N[L,J]←Z[KK+9];
COMMENT  STORE F AND SS MATRICES FOR USE IN PROCEDURE NEWLAMDA;
      IF I≥100 THEN
      BEGIN
         FL[L,J,I]←F[L,J];
         SL[L,J,I]←SS[L,J];
      END;
   END;
IF I<MAXINDEX THEN
DETM←M[1,1]×(M[2,2]×M[3,3]-M[2,3]×M[3,2])+
     M[1,2]×(M[2,3]×M[3,1]-M[2,1]×M[3,3])+
     M[1,3]×(M[2,1]×M[3,2]-M[2,2]×M[3,1]);

COMMENT  CHECK FOR CHANGE IN SIGN OF DETERMINANT OF M;

IF I<(MAXINDEX-1) AND
   SIGN(OLDDETM)≠SIGN(DETM) THEN
   IF TSING=0 THEN  BEGIN
      TSING←TB+HH×5;
      WRITE(<"THE DETERMINANT OF M CHANGED SIGN AT T=",
              F20.10>,TB); GO TO EN
   END ELSE GO TO EN;
```

```
        A←HU[I]/HUU[I];
        LOOPJ BEGIN
          CC[J]←-FU[J,I]×A;
           D[J]←HXU[J] × A;
        END;
        OLDDETM←DETM;
        LOOPJ LOOPL BEGIN
          FM[J,L]←0;
          QN[J,L]←0;
          SM[J,L]←0;
         FTN[J,L]←0;
           FOR KK←1,2,3 DO BEGIN
                          FM[J,L]←FM[J,L]+ F[J,KK]× M[KK,L];
                          QN[J,L]←QN[J,L]+ Q[J,KK]× N[KK,L];
                          FTN[J,L]←FTN[J,L]+F[KK,J]×N[KK,L];
                          SM[J,L]←SM[J,L]+SS[J,KK]× M[KK,L];
           END; END;
        KK←0;  LOOPL  MTD[L]←NTC[L]←DZ[L+21]←0;
        LOOPL  LOOPJ  BEGIN
           DZ[(KK←KK+1)]←-FM[L,J]+QN[L,J];
           DZ[KK+9]← SM[L,J] + FTN[L,J];
           DZ[(E←L+21)]←DZ[E] + FX[J,L]×Z[J+21];
           MTD[L]←MTD[L] + M[J,L]×D[J];
           NTC[L]←NTC[L] + N[J,L]×CC[J];
        END;
        LOOPL             DZ[L+18]←-MTD[L] + NTC[L];
EN: END BACK;
PROCEDURE CONTROL(T,X,DX);
```

```
    VALUE T;  REAL T;  REAL ARRAY X,DX[1];
    BEGIN
       LABEL L1,L2,L3,L4;

       IF T≥ONCE×.9999999 THEN
       BEGIN
          IF BOUND=0 THEN
          BEGIN
             IND←ENTIER(ONCE/HH+.0000001);
             GO TO L2
          END;
L1:       IND←ENTIER(ONCE/HH+.0000001)+1;
          ONCE←ONCE+HH;
L2:       BKIND←MAXINDEX-IND;IF IND≥MAXINDEX - 7  THEN GO TO L4;
          KK←0;
          LOOPI BEGIN
             B[I]←ZZ[I+18,BKIND];
             LOOPJ BEGIN
                MINV[I,J]←M[I,J]←ZZ[(KK←KK+1),BKIND];
                N[I,J]←ZZ[KK+9,BKIND]
             END;
          END;
          INVERT(MINV,3,I);
          IF I=1 THEN WRITE(<"M IS SINGULAR AT T = ",F6.2>,T);
COMMENT  COMPUTE INITIAL CONDITIONS FOR PROCEDURE NEWLAMDA;
          IF IND=100 THEN
          LOOPI BEGIN
             LOOPJ BEGIN
```

```
            TEMP4[I,J]←MINV[I,J];
            TEMP5[I,J]←    N[I,J];
         END;
         TEMP6[I]←B[I];
      END;
      IF IND=102 THEN
      BEGIN
         DELX[1]←XP[1,100]-OLDXP[1,100];
         DELX[2]←XP[3,100]-OLDXP[3,100];
         DELX[3]←XP[4,100]-OLDXP[4,100];
         LOOPI BEGIN
            SUM←0;
            LOOPJ SUM←SUM+TEMP5[J,I]×DELX[J];
            TEMP1[I]←SUM+TEMP6[I];
         END;
         LOOPI BEGIN
            SUM←0;
            LOOPJ SUM←SUM+TEMP4[J,I]×TEMP1[J];
            ZP[I,0]←SUM;
         END;
      END;
COMMENT  COMPUTE COEFFICIENTS FOR COMPUTING NEW CONTROL;
   LOOPI BEGIN
      SUM←0;
      LOOPJ SUM←SUM+FU[J,IND]×MINV[I,J];
      TEMP1[I]←SUM;
   END;
   LOOPI BEGIN
```

```
            SUM←0;
            LOOPJ SUM←SUM+TEMP1[J]×N[I,J];
            TEMP2[I]←SUM;
        END;
L4:         IF IND >MAXINDEX THEN GO TO L3;
        TEMP3[1]←OLDXP[1,IND];
        TEMP3[2]←OLDXP[3,IND];
        TEMP3[3]←OLDXP[4,IND];
        T3←T4←0;
        LOOPJ BEGIN
            CO[J,IND]←-(HUX[J,IND]+TEMP2[J])/HUU[IND];
            T3←T3+TEMP1[J]×ZZ[J+18,BKIND];
        END;
        T3←(T3+HU[IND])/HUU[IND];
        LOOPJ  T4←T4+CO[J,IND]×TEMP3[J];
        CO[4,IND]←PHI[IND]-T4-T3;
        WRITE(<"INDEX = ",I4>,IND);
        WRITE(<"C1 = ",E20.10,"C2 = ",E20.10,"C3 = ",E20.10,
       "C4 = ",E20.10>, FOR I←1 STEP 1 UNTIL 4 DO CO[I,IND]);
        WRITE(T3,T4);
        IF BOUND=0 THEN
        BEGIN
            BOUND←BOUND+1;
            GO TO L1
        END;
        END;
L3:     I ← ENTIER(T/HH);  J ← IF I<MAXINDEX THEN I+1 ELSE I;
        IF T≠OLDT THEN
```

```
        BEGIN
           OLDT←T;
           INT←T/HH-I;
           C1←(Q←CO[1,I])+INT×(CO[1,J]-Q);
           C2←(Q←CO[2,I])+INT×(CO[2,J]-Q);
           C3←(Q←CO[3,I])+INT×(CO[3,J]-Q);
           C4←(Q←CO[4,I])+INT×(CO[4,J]-Q);
        END;
        FEE←PHI[T/HH]←C1×X[1]+C2×X[3]+C3×X[4]+C4;
        R←X[1]+RE;  MASRAT←1-TOVERWO×T/ISP;
        S←SIN(X[4]);  C←COS(X[4]);  V←X[3];
        DX[1]← V×S;
        DX[2]← V×C/R;
        DX[3]← ( Q←GO×TOVERWO/MASRAT)×COS(FEE) - K×S/R*2;
        DX[4]←-K/V×C/R*2 +  Q×SIN(FEE)/V + DX[2];
END CONTROL;

COMMENT  INTEGRATION PROCEDURE GOES HERE;
      PROCEDURE NEWLAMDA(T,LAM,DLAM);
        VALUE T; REAL T; REAL ARRAY LAM,DLAM[1];
        BEGIN
           REAL A,INT,Q;
           REAL ARRAY D,DELX,SDX  ,FTDL[1:3],SDELX[1:3,100:110]
                      ,FLI[1:3,1:3];
           INTEGER IND,KK,JJ;
           LABEL L1,L2,L3;
           IF T≥ONCE×.999999999 THEN
           BEGIN
```

```
            IF BOUND=0 THEN
            BEGIN
               IND←ENTIER(ONCE/HH+.0000001);
               GO TO L2
            END;
L1:         IND←ENTIER(ONCE/HH+.0000001)+1;
            ONCE←ONCE+HH;  IF IND>MAXINDEX THEN GO TO L3;
L2:         DELX[1]←XP[1,IND]-OLDXP[1,IND];
            DELX[2]←XP[3,IND]-OLDXP[3,IND];
            DELX[3]←XP[4,IND]-OLDXP[4,IND];
            A←HU[IND]/HUU[IND];
            LOOPI BEGIN
               D[I]←HUX[I,IND]×A;
               SDELX[I,IND]←0;
               LOOPJ SDELX[I,IND]←SDELX[I,IND]+SL[I,J,IND]×
                     DELX[J];
               SDELX[I,IND]←SDELX[I,IND]-D[I];
            END;
            IF BOUND=0 THEN
            BEGIN
               BOUND←BOUND+1;
               GO TO L1
            END;
         END;
L3:      KK←ENTIER(T/HH); JJ←IF KK<MAXINDEX THEN KK+1 ELSE KK;
         INT←T/HH-KK;
         LOOPI BEGIN
            SDX[I]←(Q←SDELX[I,KK])+INT×(SDELX[I,JJ]-Q);
```

```
         LOOPJ FLI[I,J]←(Q←FL[I,J,KK])+INT×(FL[I,J,JJ]-Q);
      END;
      LOOPI BEGIN
         FTDL[I]←0;
         LOOPJ FTDL[I]←FTDL[I]+FLI[J,I]   ×LAM[J];
         DLAM[I]←  -SDX[I]-FTDL[I];
      END;
   END NEWLAMDA;

COMMENT MAIN PROGRAM BEGINS HERE;
    RE←2.09@7;  G0←32.17;  K←G0×RE*2;  ITER←0;  ISP←300;
    OLDT←6900;
    READ(XP[3,0],XP[4,0],TF,TOVERWO,NU1,NU2,HH,HUK);
    XP[1,0]←XP[2,0]←0;
    XP[4,0]←XP[4,0]/57.2957795131;
    MAXINDEX←ENTIER(TF/HH+.51);
    READ(HISTORY,FOR I←0 STEP 1 UNTIL MAXINDEX DO PHI[I]);
   WRITE(HISTORY,FOR I←0 STEP 1 UNTIL MAXINDEX DO PHI[I]);
COMMENT  INTEGRATE SYSTEM EQUATIONS WITH NOMINAL CONTROL;
    ADAMS(4,HH, 0,  TF, HH, TF, 0, 0, XP, NOMINAL);
    VO←SQRT(K/(XP[1,MAXINDEX]+RE));
    NU2←NU2/(2×VO);
    TE1←-XP[4,MAXINDEX];
    TE2←-VO  +XP[3,MAXINDEX]  ;
L1: IF ITER≠0 AND ABS(OLDTE1)<ABS(TE1) AND ABS(OLDTE2)<ABS(TE2)
    AND OLDXP[1,MAXINDEX]>XP[1,MAXINDEX] THEN
    BEGIN
       TE1←TE1/2;   TE2←TE2/2;   HUK←HUK×10;
```

```
    END ELSE HUK←HUK/2;
COMMENT   COMPUTE INITIAL CONDITIONS FOR BACKWARD INTEGRATION;
    R←XP[1,MAXINDEX]+RE;  V←XP[3,MAXINDEX];
    FOR I←2 STEP 1 UNTIL 18 DO ZZ[I,0]←0;
    ZZ[1,0]←-2×V;
    ZZ[4,0] ←  K/R*2;
    ZZ[10,0] ← -4×V×K×NU2/R*3;
    ZZ[12,0] ← -ZZ[4,0];
    ZZ[13,0] ← -2×K×NU2/R*2;
    ZZ[15,0] ←+ZZ[1,0];
    ZZ[17,0] ← 1;
    ZZ[19,0]←      -TE1;
    ZZ[20,0]←    -TE2;
    ZZ[21,0]←0;
    IF ITER=0 THEN BEGIN
    ZZ[22,0]←1+K×NU2/R*2;
    ZZ[23,0]← 2×V×NU2;
    ZZ[24,0]←-NU1;
    END ELSE BEGIN
              BOUND←0;
              ONCE←200;
              ADAMS(3,HH,200,TF,TF,TF,@-4,@-4,ZP,NEWLAMDA);
              ZZ[22,0]←ZP[1,1]+ZZ[22,0];
              ZZ[23,0]←ZP[2,1]+ZZ[23,0];
              ZZ[24,0]←ZP[3,1]+ZZ[24,0];
    END;
    TSING←0;  ONCE←0;
    DETM←0;
```

```
COMMENT  INTEGRATE M,N,B, AND LAMBDA EQUATIONS BACKWARD;
    ADAMS(24, HH, 0, TF, HH, 10, 0  ,0  , ZZ,BACK);
COMMENT  STORE OLD VALUES OF STATES BEFORE RUNNING NEW TRAJECTORY;
    FOR I←0 STEP 1 UNTIL MAXINDEX DO
    FOR L←1,2,3,4 DO OLDXP[L,I]←XP[L,I];
COMMENT  IF DET(M) CHANGED SIGN THEN A CONJUGATE POINT EXISTS
OR IF HUU WENT POSITIVE DURING THE INTEGRATION THE LEGENDRE
CONDITION IS NOT SATISFIED.  TSING IS SET TO THE TIME AT
WHICH EITHER OCCURRED AND THE FORWARD INTEGRATION IS STARTED
AT THIS POINT RATHER THAN ZERO;

    ONCE←0;
    BOUND←0;
    IF TSING=0 THEN ADAMS(4,HH,0,TF,HH,TF,0,0,XP,CONTROL)
                ELSE BEGIN
                    IF TSING>TF THEN BEGIN
                      WRITE(<"HUU WENT POSITIVE OR A CONJUGA",
                      "TE POINT OCCURRED TOO CLOSE TO THE ",
                      "END OF THE TRAJECTORY">);GO TO L2 END;
                    ONCE←ENTIER(TSING+.1);L←ENTIER(TSING/HH+.1);
                    FOR J←1,2,3,4 DO YP[J,0]←XP[J,L];
                    ADAMS(4.HH,TSING,TF,HH,HH,0,0,YP,CONTROL);
                    FOR I←L STEP 1 UNTIL MAXINDEX DO
                    FOR J←1,2,3,4 DO XP[J,I]←YP[J,I-L];
                END;
 WRITE(FOR I←0 STEP 1 UNTIL MAXINDEX DO PHI[I]);
    VO←SQRT(K/(XP[1,MAXINDEX]+RE));
```

```
    OLDTE1←TE1;   OLDTE2←TE2;
    TE1←-XP[4,MAXINDEX];
    TE2←-VO  +XP[3,MAXINDEX]  ;
COMMENT  IF IMPROVEMENT IN TERMINAL ERRORS AND PAYOFF IS
LESS THAN EPSILON (USERS CHOICE) THEN STOP THE PROGRAM;
    IF ABS(OLDTE1-TE1)≤.001AND ABS(OLDTE2-TE2)≤1
    AND ABS(OLDXP[1,MAXINDEX]-XP[1,MAXINDEX])≤500   THEN
    BEGIN
       WRITE(FOR I←1 STEP 1 UNTIL MAXINDEX DO PHI[I]);
       GO TO L2
    END ELSE
    BEGIN
       ITER←ITER+1;
       WRITE(F1,NU1,NU2,TE1,TE2,HUK);
       GO TO L1
    END;
L2: END.
```

REFERENCES

1. Kelley, H. J., "Gradient Theory of Optimal Flight Paths", Journal of the American Rocket Society, vol. 30, 1960, pp. 947-953.

2. Bryson, A. E. and Denham, W. F., "A Steepest-Ascent Method for Solving Optimum Programming Problems", Journal of Applied Mechanics, vol. 29, 1962, pp. 247-257.

3. Rosenbaum, R., "Convergence Technique for the Steepest-Descent Method of Trajectory Optimization", AIAA Journal, vol. 1, 1963, pp. 1703-1705.

4. Bryson, A. E. and Denham, W. F., "Multivariable Terminal Control for Minimum Mean Square Deviation from a Nominal Path", Proceedings of the IAS Symposium on Vehicles Systems Optimization (Institute of the Aerospace Sciences, New York, 1961), pp. 91-97.

5. Breakwell, J. V., Speyer, J. L., and Bryson, A. E., "Optimization and Control of nonlinear Systems Using the Second Variation", J. Soc. Ind. Appl. Math., vol. 1, 1963, pp. 193-223.

6. Bullock, T. E. Jr., "Computation of Optimal Controls by a Method Based on Second Variations", Ph.D. Thesis, Stanford University, 1966.

7. DeRusso, P. M., Roy, R. J., and Close, C. M., State Variables for Engineers, John Wiley and Sons, Inc., New York, 1965.